http://dx.doi.org/10.5935/0103-5053.20130254

J. Braz. Chem. Soc., Vol. 24, No. 12, 2028-2032, 2013.
Printed in Brazil - ©2013 Sociedade Brasileira de Química
0103 - 5053 $6.00+0.00

Article

Bursting in the Belousov-Zhabotinsky Reaction Added with Phenol in a Batch Reactor

Ariel Cadena,[a] *Daniel Barragán*[b] *and Jesús Ágreda**[,a]

[a]*Departamento de Química, Facultad de Ciencias, Universidad Nacional de Colombia, Cra 30 No. 45-03, Bogotá, Colombia*

[b]*Escuela de Química, Facultad de Ciencias, Universidad Nacional de Colombia, Calle 59A No. 63-20, Oficina 16-413, Medellín, Colombia*

A reação de Belousov-Jabotinsky clássica foi modificada pela adição de fenol como um segundo substrato orgânico que compete cineticamente com o ácido malônico na redução de Ce^{4+} para Ce^{3+} e na remoção de bromo molecular da reação. A reação oscilante de dois substratos exibiu oscilações abruptas e período oscilatório de longa duração. A análise de dados experimentais mostra um aumento do fenômeno abrupto, com pico maior e estado quiescente mais longo, como função do aumento da concentração de fenol inicial. Hipotetizou-se que o fenômeno de oscilação abrupta pode ser explicado pela introdução de um ciclo redox entre as espécies fenólicas reduzidas (hidroxifenois) e as oxidadas onas (quinonas). A hipótese foi testada experimentalmente e numericamente e dos resultados concluiu-se que o fenômeno oscilatório abrupto exibido pela reação oscilante de dois substratos é impulsionado principalmente por um ciclo redox *p*-di-hidroxi-benzeno/*p*-benzoquinona.

The classic Belousov-Zhabotinsky reaction was modified by adding phenol as a second organic substrate that kinetically competes with the malonic acid in the reduction of Ce^{4+} to Ce^{3+} and in the removal of molecular bromine of the reaction mixture. The oscillating reaction of two substrates exhibited burst firing and an oscillatory period of long duration. Analysis of experimental data shows an increasing of the bursting phenomenon, with a greater spiking in the burst firing and with a longer quiescent state, as a function of the initial phenol concentration increase. It was hypothesized that the bursting phenomenon can be explained introducing a redox cycle between the reduced phenolic species (hydroxyphenols) and the oxidized ones (quinones). The hypothesis was experimentally and numerically tested and from the results it is possible to conclude that the bursting phenomenon exhibited by the oscillating reaction of two substrates is mainly driven by a *p*-di-hydroxy-benzene/*p*-benzoquinone redox cycle.

Keywords: reaction kinetics, mechanisms, phenolic compounds, BZ reaction, bursting phenomenon

Introduction

The Belousov-Zhabotinsky (BZ) and uncatalyzed bromate oscillator (UBO) reactions have been studied in a wide of experimental conditions in both batch and continuous stirred tank (CSTR) reactors.[1-5] Complex dynamic behaviors in oscillating reactions have been found in non-stirred batch reactors and in CSTR and electrochemical setups.[6-12] These amazing reactions have become relevant in science, and particularly in biochemistry, due to their similarity with the dynamic activity of many cellular control processes.[13] On the other hand, travelling waves, Turing patterns, burst firing, sequential oscillations and chaotic phenomena are some of the more common spatio-temporal dynamics studied in oscillating reactions.[14-18] Looking for a better understanding of the reaction mechanism and dynamics of oscillating reactions, researchers have employed several substrates, individually or mixed, in the study of BZ reaction. The induction period, frequency, amplitude, shape and periodicity of oscillations change by the presence of a new organic or inorganic substance in the reaction mixture of BZ reaction.[19-29]

In the present work, the oscillatory dynamics of the classic BZ reaction[30,31] (a mixture of malonic acid, Ce^{4+} and

*e-mail: jaagredab@unal.edu.co

bromated in sulfuric acidic) were studied in a batch reactor in the presence of phenol as a second organic substrate that kinetically competes with the malonic acid in the reduction of Ce^{4+} to Ce^{3+} and in the removal of molecular bromine. With the two substrates (malonic acid-phenol), BZ reaction shows an astonishing variation of its dynamics as a function of the initial concentration of phenol, exhibiting enhanced periods of oscillations and bursting phenomenon. At first glance, the malonic acid-phenol BZ oscillator was thought as a system of coupled oscillators because the bromate-phenol-sulfuric acidic is a well-known oscillating chemical reaction (UBO). In order to test this idea of coupled oscillators, a set of numerical simulations by using an extended reaction mechanism based on the Marburg-Budapest-Missoula (MBM)[32] and Gyorgyi, Varga, Körös, Field and Ruoff (GVKFR)[33] reaction schemes was carried out. The MBM mechanism for the cerium-catalyzed BZ reaction is a complete reaction scheme that includes both negative feedback loops and radical-radical recombination reactions of organic species.[32] Whereas the GVKFR model is a mechanism to explain the oscillations observed in the *p*-hydroxyphenol-bromate-acidic media reaction,[33] the closer and complete mechanism available in the literature to UBO that uses phenol as organic substrate. As a result of these numerical simulations, some interesting behaviors were obtained, but there was nothing to indicate the possibility of burst firing.

In order to address the experimental evidence of that burst firing obtained in this work for the malonic acid-phenol BZ reaction, a second hypothesis is that the subproducts of phenol oxidation (hydroxyphenols-quinones) are involved in a novel redox cycle coupled to the main catalytic cycle of cerium ions. A series of experiments using 1,4-benzoquinone, 2-hydroxyphenol and 4-hydroxyphenol instead of phenol as a second organic substrate were carried out to test this hypothesis. The experimental results obtained by adding benzoquinone and hydroxyphenols to the BZ reaction suggest that the hypothesized hydroxyphenol-quinone redox cycle can be accepted as plausible. This hypothesis was materialized as a set of reaction steps, and they were incorporated into an extended MBM-GVKFR mechanism. The numerical simulation results of this new model (MBM-GVKFR-hydroxyphenols-quinones redox cycle) support the idea of the hydroxyphenols-quinones redox cycle.

Experimental

Sulfuric acid (Merck 95-98% extra pure), $KBrO_3$ (Carlo Erba Milano ACS Titolo min 99.8%), $Ce(SO_4)_2 \cdot 4H_2O$ (Merck zur Analyse > 98%), malonic acid (Merck zur Synthese), phenol (JT Baker Chemicals B. V. "Baker Grade"), 2-hydroxyphenol (Fisher Scientific Company), 4-hydroxyphenol (Merck zur Synthese) and *p*-benzoquinone (Hopkin and Williams, LTD.) were used as received. All solutions were prepared in deionized water. The initial concentration of phenol used in the experiments were: a. 0.00, b. 0.05347, c. 0.1337, d. 0.2673, e. 0.5347, f. 1.069, g. 1.337, h. 1.604, i. 1.871, j. 2.272, k. 2.673, l. 3.074, m. 3.476, n. 3.877, o. 4.278, p. 4.679, q. 5.347, and r. 10.69 mmol L^{-1}. The initial concentrations of classic BZ reagents were: 28.90 mmol L^{-1} $KBrO_3$, 26.06 mmol L^{-1} malonic acid, 0.5606 mmol L^{-1} $Ce(SO_4)_2$ and 1.00 mol L^{-1} H_2SO_4. A thermostated (25.0 ± 0.1 °C) 100 mL double jacket cylindrical cell, with magnetic stirring at 500 rpm, was used to obtain the potentiometric measurements, using a platinum electrode Mettler-Toledo Pt4805-60-88TE-S7/120 combination ORP/Redox with Ag/AgCl reference (movable PTFE reference junction). All the experiments were made at least by duplicate.

Results and Discussion

The malonic acid-phenol BZ reaction exhibits a striking alteration of its temporal oscillatory dynamics as a function of the initial concentration of phenol. An enlargement of the oscillatory regime and the onset of bursting phenomenon are the more important observed effects by the addition of phenol to BZ reaction. The length of the induction time, the amplitude of sustained oscillations and the increasing of the total oscillatory reaction time are closely correlated with the initial concentration of phenol, and the burst firing appears when the malonic acid-phenol concentration ratio ranges between 25 and 6. Figure 1 shows the temporal redox potentiometric measurements of BZ reaction in the presence of an initial concentration of phenol (curves a to r). The BZ reaction (Figure 1 curve a) has an oscillatory reaction time of around 2 h, while at the same initial concentrations but in the presence of 3.074 mmol L^{-1} phenol (Figure 1 curve l), the oscillatory reaction time extends to almost 30 h.

Figure 1 also shows other highlighting features of this BZ oscillating reaction of two substrates. When the initial concentration of phenol is lower than 2.0 mmol L^{-1} (Figure 1 curves from a to i), the reaction mixture exhibits sustained oscillatory phase, and the period and the amplitude of the oscillations remain constant before a sudden ending. If the initial concentration of phenol, ranges between 2.0 and 10 mmol L^{-1} (Figure 1 curves j to q), the BZ reaction mixture shows a transition from sustained to damped oscillations during its temporal evolution. But the most astonishing observed effect of phenol on the dynamics

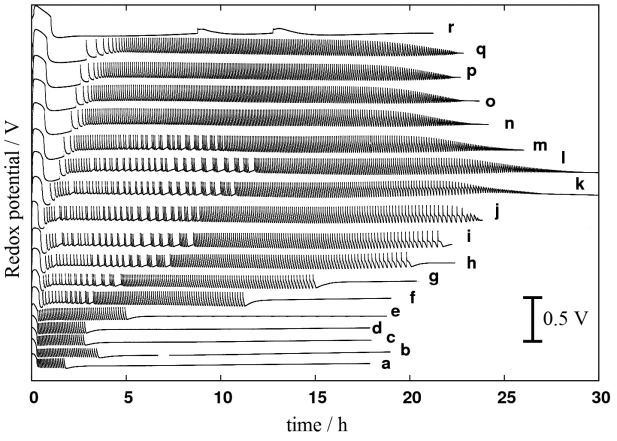

Figure 1. Redox potentiometric signal against time for the BZ oscillating reaction of two substrates. Initial concentrations for BZ reagents and initial concentration of phenol (curves a to r) are given in the experimental section.

of BZ reaction was the bursting phenomenon. If the initial concentration of phenol ranges between 1.069 and 3.476 mmol L^{-1} (Figure 1 curves f to m), the reaction mixture exhibits a complex temporal transition from burst firing and sustained oscillations (ended suddenly), to still burst firing but damped oscillations. This means that the malonic acid-phenol BZ reaction in a batch reactor evolves in time through different attractors: period-n bursting attractor, limit cycle and stable focus.

In order to explain the experimental results showed in Figure 1, a new redox cycle is propose: the Ce^{4+} oxidation of phenol to *p*-quinones[34-37] followed by the reduction of *p*-quinones to phenolic compounds mediated by transient reactive organic free radicals in solution,[38] like carboxyl ($COOH^{\bullet}$) or tartronyl ($TA^{\bullet}$).[31,32,39] If this aromatic redox cycle is plausible, then the Ce^{4+}/Ce^{3+} catalytic cycle of BZ reaction is involved in a kinetic competition, the reduction of $BrO_2^{\bullet}$ by Ce^{3+} or phenol. In a typical antioxidation action, common in phenols, the oxidation of Ce^{3+} is diminished, and because of this, the consumption of malonic acid is slower, whereas the phenol consumption is higher. These facts together increase the oscillation time of the BZ reaction of two substrates. Now, at low concentration of phenolic compounds, the Ce^{4+} concentration rises, and the Ce^{4+}/Ce^{3+} catalytic cycle drives the BZ reaction while quinone type compounds are reduced to phenolic compounds by some reactive free radicals, like the carboxyl ($COOH^{\bullet}$) or the tartronyl ($TA^{\bullet}$) radicals. When the concentration of phenolic compounds increases, the aromatic cycle (phenol-quinone) restarts and drives the oscillating reaction. In this way, it is proposed that the two catalytic cycles alternate to drive the reaction until the oscillatory period ends. It is important to take into account that the polymerization of the quinones takes place at the reaction mixture conditions, as it is well-known from the UBO chemical oscillator.[33,34] The polymers of quinone are almost insoluble and their reduction by free radicals is not a viable process and we suppose that they are involved in the burst firing by way of a non-synchronized action between catalytic processes. It is also important to remark that at high enough concentration of phenol, the Ce^{4+} oxidation of phenolic compounds is a kinetically preferred process, instead of the Ce^{4+} oxidation of aliphatic species of the BZ reaction.[40]

All the above ideas have the aim to help to understand, from a mechanistic point of view, the complex behavior exhibited by the malonic acid-phenol BZ oscillating reaction, and those ideas are summarized in the next way: at the beginning of reaction, the oxidation of phenol by Ce^{4+} is the kinetically favored process with a slow consumption of bromate and malonic acid; in this way, whereas the concentration of phenolic compounds is over a critical value, the aromatic catalytic cycle drives BZ oscillations; and when the phenol concentration is high enough, insoluble polymers of quinone are produced and an irregular oscillatory dynamic appears, the bursting phenomenon. At long times, the phenolic compounds are decreased in the reaction mixture, and a sequential train of sustained oscillations, driven by the consumption of malonic acid by the Ce^{4+}/Ce^{3+} catalytic cycle, leads the oscillations to its end.

In order to get some experimental clues about the participation of phenol in the malonic acid BZ reaction, the classic experiment was tested in the presence of some key aromatic compounds, 2-hydroxyphenol, 4-hydroxyphenol and *p*-benzoquinone. The results are in Figure 2, in which curves c and e show that the BZ reaction of two substrates (malonic acid with *p*-benzoquinone and malonic acid with *4*-hydroxyphenol, respectively) exhibits a dynamic behavior of a similar type that the malonic acid with phenol, curve b. On the contrary, the *2*-hydroxyphenol, curve d, does not modify, to an appreciable extent, the malonic acid BZ reaction, curve a. The main result in Figure 2 is the evidence suggesting that the proposal, about the oxidation-reduction cycle, among phenol and quinone compounds, is capable to explain the experimental results. Also, it says that the main compound in the phenol-quinone process is a *para* compound and not an *orto* compound, as could be inferred from UBO oscillators.[33]

Numerical simulations

Numerical simulations were used as a tool to treat the fully nonlinear dynamics of the complex chemical BZ reaction and to test the validity of the previously presented hypothesis. The following set of reactions 1 to 9

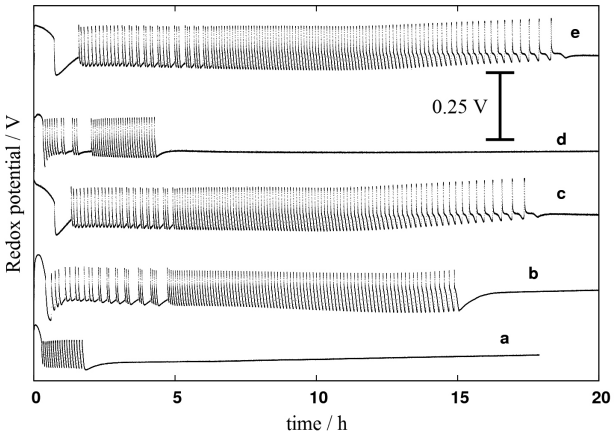

Figure 2. Redox potentiometric signal against time for the BZ oscillating reaction of two substrates. a: malonic acid alone, b: malonic acid with phenol, c: malonic acid with 1,4-benzoquinone, d: malonic acid with 2-hydroxyphenol, and e: malonic acid with 4-hydroxyphenol. The phenolic species were added in concentration of 1.3 mmol L^{-1}. The other concentrations were the same as in Figure 1.

was added to the complete set of reactions of MBM plus the GVKFR mechanisms (the complete set of reaction rates, kinetic constants and the fortran source code used for these simulations are presented in the Supplementary Information (SI) section).

Phenol oxidation reactions:

$$Ce^{4+} + C_6H_6O \underset{k_r = 4\,x\,10^4}{\overset{k_r = 1\,x\,10^3}{\longleftrightarrow}} Ce^{3+} + C_6H_5O^{\bullet} + H^+ \quad (1)$$

$$Ce^{4+} + C_6H_5O^{\bullet} + H_2O \underset{k_r = 0}{\overset{k_r = 1\,x\,10^4}{\longleftrightarrow}} Ce^{3+} + C_6H_6O_2 + H^+ \quad (2)$$

$$Ce^{4+} + C_6H_6O_2 \underset{k_r = 4\,x\,10^4}{\overset{k_r = 1\,x\,10^3}{\longleftrightarrow}} Ce^{3+} + C_6H_5O_2^{\bullet} + H^+ \quad (3)$$

$$Ce^{4+} + C_6H_5O_2^{\bullet} \underset{k_r = 0}{\overset{k_r = 1\,x\,10^4}{\longleftrightarrow}} Ce^{3+} + C_6H_4O_2 + H^+ \quad (4)$$

Quinone reduction reactions:

$$COOH^{\bullet} + C_6H_4O_2 \underset{k_r = 0}{\overset{k_r = 1\,x\,10^6}{\longleftrightarrow}} CO_2 + C_6H_5O_2^{\bullet} \quad (5)$$

$$COOH^{\bullet} + C_6H_5O_2^{\bullet} \underset{k_r = 0}{\overset{k_r = 1\,x\,10^8}{\longleftrightarrow}} CO_2 + C_6H_6O_2 \quad (6)$$

$$TA^{\bullet} + C_6H_4O_2 \underset{k_r = 0}{\overset{k_r = 1\,x\,10^6}{\longleftrightarrow}} MOA + C_6H_5O_2^{\bullet} \quad (7)$$

$$TA^{\bullet} + C_6H_5O_2^{\bullet} \underset{k_r = 0}{\overset{k_r = 1\,x\,10^7}{\longleftrightarrow}} MOA + C_6H_6O_2 \quad (8)$$

Quinone consumption:

$$C_6H_4O_2 \underset{k_r = 0}{\overset{k_r = 1\,x\,10^5}{\longleftrightarrow}} \text{oxidated products} \quad (9)$$

In these reactions, TA is for tartronic acid and MOA is for meso-oxalic acid.[30-32] The rate constants were estimated based on similar reactions of the MBM and GVKFR mechanisms. Reactions 1 to 4 describe a sequential electron transfer for the Ce^{4+} and the resulting oxidation of phenol to the corresponding quinone. Reactions 5 to 8 indicate a plausible sequential reduction of quinone, by reactive organic free radicals, to phenolic like compounds. The selection of free radical species involved in reactions 5 to 8 was based on redox potentials. The carboxyl radical ($COOH^{\bullet}$) has a standard redox potential of −1.82 V *vs.* NHE.[38] On the other hand, the tartronyl radical ($TA^{\bullet}$) was chosen as a representative free radical that has been found in the BZ reaction (like the malonyl and bromomalonyl free radicals).[30-32] Finally, the reaction 9 describes the irreversible degradation or polymerization of quinones. Figure 3 shows the results obtained for simulations.

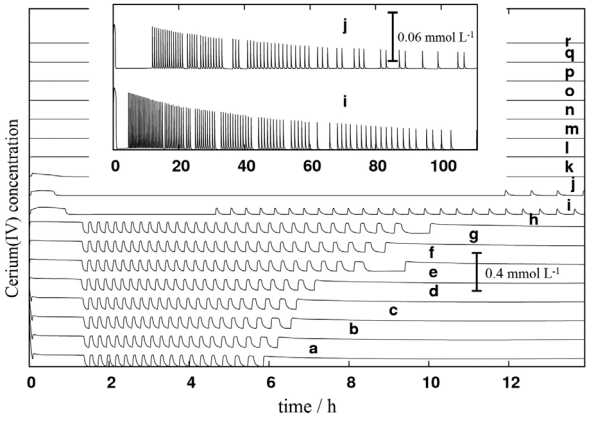

Figure 3. Numerical simulations of the two substrates, malonic acid and phenol, BZ reaction. Curves a to r are for the same initial concentrations used for Figure 1.

The numerical simulations have some of the experimental observed characteristics of the malonic acid-phenol BZ oscillating reaction, like an induction time enlargement and an increasing oscillatory reaction time as the initial phenol concentration increases. Also, the burst firing appears, and it is the most interesting result (inset in Figure 3). This qualitative agreement, between the experiments and the numerical simulations, is in favor of the hypothesized phenol-quinone redox cycle. However, it is necessary to confirm these ideas, in future works, by determining the experimental rate constant values, and by including, or deleting, some reactions. Also, a chromatographic and electron paramagnetic resonance (EPR) spectroscopy studies would be particularly useful to find the specific intermediaries.

Conclusions

The results presented in this work show the dynamic behavior of the malonic acid-phenol BZ reaction. It is interesting the appearance of bursting phenomenon in a

closed system. The burst firing origin was explained as a complex process that involves a kinetic competition between an aromatic redox cycle of phenolic compounds and the Ce^{4+}/Ce^{3+} catalytic cycle of the BZ classic oscillator. In this way, the presence of phenol in the malonic acid BZ reaction plays a role as an antioxidant agent preventing the oxidation of the malonic acid, and its derivatives, by Ce^{4+} ions.

Supplementary Information

Supplementary data are available free of charge at http://jbcs.sbq.org.br as PDF file.

Acknowledgment

This project has been supported by the DIB of the Universidad Nacional de Colombia under the code 803638.

Reference

1. Sagués, F.; Epstein, I. R.; *Dalton Trans.* **2003**, *7*, 1201.
2. Básángi, T.; Leda, M. Jr.; Toiya, M.; Zhabotinsky, A. M.; Epstein, I. R.; *J. Phys. Chem. A* **2009**, *113*, 5644.
3. Gentili, P. L.; Horvath, V.; Vanag, V. K.; Epstein, I. R.; *Int. J. Unconv. Comput.* **2012**, *8*, 177.
4. Adamciková, L.; Misicák, D.; Sevcik, P.; *React. Kinet. Catal. Lett.* **2005**, *85*, 215.
5. Szabo, E.; Adamciková, L.; Sevcik, P.; *J. Phys. Chem. A* **2011**, *115*, 6518.
6. Ruoff, P.; *J. Phys. Chem.* **1992**, *96*, 9104.
7. Grancicova, O.; Olexova, A.; *Z. Phys. Chem.* **2009**, *223*, 1451.
8. Rachwalska, M.; Kawczynski, A. L.; *J. Phys. Chem. A* **2001**, *105*, 7885.
9. Badola, P.; Rajani, P.; Ravi-Kumar, V; Kulkarni, B. D.; *J. Phys. Chem.* **1991**, *9*, 2939.
10. Bronnikova, T. V.; Schaffer, W. M.; Olsen, L. F.; *J. Phys. Chem. B* **2001**, *105*, 310.
11. Kiss, I. Z.; Lv, Q.; Organ, L.; Hudson, J. L.; *Phys. Chem. Chem. Phys.* **2006**, *8*, 2707.
12. Simo, H.; Woafo, P.; *Mech. Res. Comm.* **2011**, *38*, 537.
13. Izhikevich, E. M.; *Int. J. Bifurcation Chaos* **2000**, *10*, 1171.
14. Lengyel, I.; Epstein, I. R.; *Acc. Chem. Res.* **1993**, *26*, 235.
15. Epstein, I. R.; Showalter, K.; *J. Phys. Chem.* **1996**, *100*, 13132.
16. Epstein, I. R.; Pojman, J. A.; *An Introduction to Nonlinear Chemical Dynamics*; Oxford University Press: New York, USA, 1998.
17. Rastogi, R. P.; *Introduction to Non-Equilibrium Physical Chemistry*; Elsevier: Amsterdam, The Netherlands, 2008.
18. Pojman, J. A.; Tran-Cong-Miyata, Q.; *Nonlinear Dynamics with Polymers*; Wiley-VCH Velarg: Weinheim, Germany, 2010.
19. Heilweil, E. J.; Henchman, M. J.; Epstein, I. R.; *J. Am. Chem. Soc.* **1979**, *101*, 3698.
20. Treindl, L.; Ruoff, P.; Kvernberg, P. O.; *J. Phys. Chem. A* **1997**, *101*, 4606.
21. Rastogi, R. P.; Chand, P.; Pandey, M. K.; Das, M.; *J. Phys. Chem. A* **2005**, *109*, 4562.
22. Shah, S.; Wang, J.; *J. Phys. Chem. C* **2007**, *111*, 10639.
23. Zeyer, K.-P.; Schneider, F. W.; *J. Phys. Chem. A* **1998**, *102*, 9702.
24. Chen, Y.; Wang, J.; *J. Phys. Chem. A* **2005**, *109*, 3950.
25. Amemiya, T.; Wang, J.; *J. Phys. Chem. A* **2010**, *114*, 13347.
26. Schwarz, H. A.; Dodson, R. W.; *J. Phys. Chem.* **1989**, *93*, 409.
27. Li, N.; Wang, J.; *J. Phys. Chem. A* **2008**, *112*, 6281.
28. Biosa, G.; Ristori, S.; Spalla, O.; Rustici, M.; Hauser, M. J. B.; *J. Phys. Chem. A.* **2011**, *115*, 3227.
29. Asakura, K.; Konishi, R.; Nakatani, T.; Nakano, T.; Kamata, M.; *J. Phys. Chem. B* **2011**, *115*, 3959.
30. Gyorgyi, L.; Turányi. T.; Field, R. J.; *J. Phys. Chem.* **1990**, *94*, 7162.
31. Gyorgyi, L.; Turányi, T.; Field, R. J.; *J. Phys. Chem.* **1993**, *97*, 1931.
32. Hegedeus, L.; Wittman, M.; Noszticzius, Z.; Yan, S.; Sirimungkala, A.; Försterling, H. D.; Field, R. J.; *Faraday Discuss.* **2002**, *120*, 21.
33. Györgyi, L.; Varga, M.; Körös, E.; Field, R. J.; Ruoff, P.; *J. Phys. Chem.* **1989**, *93*, 2836.
34. Spence, W. R.; Duke, F. R.; *Anal. Chem.* **1954**, *26*, 919.
35. Dixon, W. T.; Murphy, D.; *J. Chem. Soc., Perkin Trans.* **1975**, *2*, 850.
36. Domagała, S.; Steglińska, V.; Dziegieć, J.; *Monatsh. Chem. Chem. Mon.* **1998**, *129*, 761.
37. Simon, A.; Ballai, C.; Lente, G.; Fábián, I.; *New. J. Chem.* **2011**, *35*, 235.
38. Wardman, P.; *Reduction Potentials of One-Electron Couples Involving Free Radicals in Aqueous Solution*; American Chemical Society and the American Institute of Physics for the National Institute of Standards and Technology: USA, 1989, p. 1637.
39. Blagojevic, S. M.; Anic, S. R.; Cupic, Z. D.; *Russ. J. Phys. Chem. A* **2011**, *85*, 2274.
40. Singh, M. P.; Singh, H. S.; Verma, M.; *J. Phys. Chem.* **1980**, *84*, 256.

Submitted: July 27, 2013
Published online: October 16, 2013

J. Braz. Chem. Soc., Vol. 24, No. 12, S1-S71, 2013.
Printed in Brazil - ©2013 Sociedade Brasileira de Química
0103 - 5053 $6.00+0.00

Supplementary Information

Bursting in the Belousov-Zhabotinsky Reaction Added with Phenol in a Batch Reactor

Ariel Cadena,[a] Daniel Barragán[b] and Jesús Ágreda*[,a]

[a]Departamento de Química, Facultad de Ciencias, Universidad Nacional de Colombia, Cra 30 No. 45-03, Bogotá, Colombia

[b]Escuela de Química, Facultad de Ciencias, Universidad Nacional de Colombia, Calle 59A No. 63-20, Oficina 16-413, Medellín, Colombia

Table S1. Species nomenclature and initial concentrations used for the simulation of the BZ reaction added with phenol

Specie representation	Initial concentration used for simulations / (mol L^{-1})	Specie name	Mechanism
c(1)	0	Br^-	MBM and GVKFR
c(2)	0	HOBr	MBM and GVKFR
c(3)	1.29	H^+	MBM and GVKFR
c(4)	0	Br_2	MBM and GVKFR
c(5)	0	$HBrO_2$	MBM and GVKFR
c(6)	2.8896×10^{-2}	BrO_3^-	MBM and GVKFR
c(7)	0	$H_2BrO_2^+$	MBM and GVKFR
c(8)	0	Br_2O_4	MBM and GVKFR
c(9)	0	BrO_2*	MBM and GVKFR
c(10)	0	Ce^{+3}	MBM
c(11)	5.606×10^{-4}	Ce^{+4}	MBM
c(12)	0	O_2	MBM
c(13)	0	BrMA[b]	MBM
c(14)	0	BrMA*[b]	MBM
c(15)	0	BrEETRA[b]	MBM
c(16)	0	CO_2	MBM
c(17)	0	BrMA(enol)[b]	MBM
c(18)	0	Br_2MA[b]	MBM
c(19)	0	BrMABrO_2[b]	MBM
c(20)	0	OA[b]	MBM
c(21)	0	BrTA[b]	MBM
c(22)	0	MOA[b]	MBM
c(23)	0	COOH*[b]	MBM
c(24)	0	MA*[b]	MBM
c(25)	2.6056×10^{-2}	MA[b]	MBM
c(26)	0	ETA[b]	MBM

*e-mail: jaagredab@unal.edu.co

Table S1. continuation

Specie representation	Initial concentration used for simulations / (mol L^{-1})	Specie name	Mechanism
c(27)	0	MA(enol)[b]	MBM
c(28)	0	$MABrO_2$[b]	MBM
c(29)	0	TA[b]	MBM
c(30)	0	EETA[b]	MBM
c(31)	0	TA*[b]	MBM
c(32)	0	EEHTRA[b]	MBM
c(33)	0	TA(enol)[b]	MBM
c(34)	0	$TABrO_2$[b]	MBM
c(35)	variable[a]	Fenol	GVKFR
c(36)	0	Fenol*	GVKFR
c(37)	0	Rox1[c]	GVKFR
c(38)	0	RBr[c]	GVKFR
c(39)	0	RBr_2[c]	GVKFR
c(40)	0	RBr*[c]	GVKFR
c(41)	0	R(BrOH)[c]	GVKFR
c(42)	0	Rox2[c]	GVKFR
c(43)	0	OQN[c]	GVKFR
c(44)	0	RBr_2*[c]	GVKFR
c(45)	0	RBr(BrOH)[c]	GVKFR
c(46)	0	Rox3[c]	GVKFR
c(47)	0	BrOQN[c]	GVKFR
c(48)	0	Rox4[c]	GVKFR
c(49)	0	*p*-hidroxyphenol	proposed in this work
c(50)	0	*p*-hidroxyphenol radical	proposed in this work
c(51)	0	*p*-quinone	proposed in this work

[a]Everyone of the 18 experimental concentrations of phenol was used here. [b]The nomenclature use in references 32 of the paper was used. [c]The nomenclature use in references 33 of the paper was used.

Table S2. The complete set of reaction rates used for the simulation of the bursting phenomena in the BZ reaction added with phenol. The numerical values are the respective kinetic constants. The negative sign precedes the reverse reaction when the reverse kinetic constant is different to zero

Reaction	Reaction rate	Mechanism
V(1)	8×10^9 c(1) c(2) c(3) - 110 c(4)	MBM and GVKFR
V(2)	2.9×10^6 c(1) c(5) c(3) - 2×10^{-5} c(2) c(2)	MBM and GVKFR
V(3)	0.6 c(1) c(6) c(3) c(3) - 3.2 c(2) c(5)	MBM and GVKFR
V(4)	2×10^6 c(5) c(3) - 1.0×10^8 c(7)	MBM and GVKFR
V(5)	1.7×10^5 c(5) c(7)	MBM and GVKFR
V(6)	48 c(5) c(6) c(3) - 3.2×10^3 c(8)	MBM and GVKFR
V(7)	7.5×10^4 c(8) - 1.4×10^9 c(9) c(9)	MBM
V(8)	6×10^4 c(10) c(9) c(3) - 1.3×10^4 c(11) c(5)	MBM
V(9)	6×10^{-10} c(6) c(6) c(3) c(3)	MBM
V(10)	0.06 c(9)	MBM
V(11)	0.1 c(13) c(11) - 400 c(14) c(10) c(3)	MBM
V(12)	1×10^9 c(14) c(14)	MBM
V(13)	1.2×10^{-2} c(13) - 800 c(17)	MBM
V(14)	3.5×10^6 c(17) c(4)	MBM
V(15)	1.1×10^6 c(17) c(2)	MBM
V(16)	2×10^9 c(14) c(9)	MBM
V(17)	0.62 c(19)	MBM
V(18)	0.46 c(19)	MBM
V(19)	1.5 c(21)	MBM
V(20)	7×10^3 c(22) c(11)	MBM
V(21)	28 c(20) c(11)	MBM
V(22)	5×10^9 c(23) c(23)	MBM
V(23)	1×10^{97} c(23) c(11)	MBM
V(24)	0.6×10^6 c(23) c(13)	MBM
V(25)	3×10^9 c(23) c(14)	MBM
V(26)	5×10^9 c(23) c(9)	MBM
V(27)	0.23 c(25) c(11) - 2.2×10^4 c(24) c(10) c(3)	MBM
V(28)	3.2×10^9 c(24) c(24)	MBM
V(29)	2.6×10^{-3} c(25) - 180 c(27)	MBM
V(30)	2×10^6 c(27) c(4)	MBM
V(31)	6.7×10^5 c(27) c(2)	MBM
V(32)	5×10^9 c(24) c(9)	MBM
V(33)	0.55 c(28)	MBM
V(34)	1.0 c(28)	MBM
V(35)	2×10^9 c(24) c(14)	MBM
V(36)	4×10^9 c(24) c(23)	MBM
V(37)	0.66 c(29) c(11) - 1.7×10^4 c(31) c(10) c(3)	MBM
V(38)	1×10^9 c(31) c(31)	MBM
V(39)	2.3×10^{-5} c(29) - 1.5 c(33)	MBM
V(40)	3×10^5 c(33) c(4)	MBM
V(41)	2×10^5 c(33) c(2)	MBM
V(42)	1×10^9 c(31) c(24)	MBM
V(43)	1×10^9 c(31) c(14)	MBM

Table S2. continuation

Reaction	Reaction rate	Mechanism
V(44)	3×10^{9} c(31) c(23)	MBM
V(45)	2×10^{9} c(31) c(9)	MBM
V(46)	0.1 c(34)	MBM
V(47)	5×10^{-5} c(29) c(6)	MBM
V(48)	160 c(24) c(6) c(3)	MBM
V(49)	1×10^{-2} c(35) c(6) c(3)	GVKFR
V(50)	1×10^{4} c(35) c(9)	GVKFR
V(51)	1×10^{4} c(36) c(6) c(3)	GVKFR
V(52)	6×10^{5} c(35) c(4)	GVKFR
V(53)	2×10^{3} c(38) c(4)	GVKFR
V(54)	1×10^{4} c(38) c(9)	GVKFR
V(55)	1×10^{4} c(40) c(6) c(3)	GVKFR
V(56)	1×10^{4} c(40) c(6) c(3)	GVKFR
V(57)	2×10^{-1} c(41)	GVKFR
V(58)	1×10^{4} c(39) c(9)	GVKFR
V(59)	1×10^{4} c(44) c(6) c(3)	GVKFR
V(60)	2.5×10^{3} c(44) c(6) c(3)	GVKFR
V(61)	2×10^{-1} c(45)	GVKFR
V(62)	1×10^{2} c(4) c(37)	GVKFR
V(63)	1×10^{3} c(11) c(35) - 4×10^{4} c(10) c(36) c(3)	our proposal
V(64)	1×10^{4} c(36) c(11)	our proposal
V(65)	1×10^{3} c(49) c(11) - 4×10^{4} c(50) c(10) c(3)	our proposal
V(66)	1×10^{4} c(50) c(11)	our proposal
V(67)	1×10^{6} c(51) c(23)	our proposal
V(68)	1×10^{8} c(50) c(23)	our proposal
V(69)	1×10^{6} c(51) c(31)	our proposal
V(70)	1×10^{7} c(50) c(31)	our proposal
V(71)	1×10^{0} c(28)	MBM, called NR5
V(72)	1×10^{5} c(51)	our proposal

All variables used in the computer program were double precision, and the tolerance for the convergence of the algorithm was set to 1×10^{-10}. Others tolerances were tested but few changes were found and no more detailed studies were try respect to this parameter.

Source code

```
!Revisado septiembre 17 de 2013.
   Program BarridoFenol
   Implicit none
   DOUBLE PRECISION CFenol
   Integer contador
!  Do contador=1,18,1
!     If (contador .EQ. 1) then; CFenol=0D0; end if
!     If (contador .EQ. 2) then; CFenol=5.347D-5; end if
!     If (contador .EQ. 3) then; CFenol=1.337D-4; end if
!     If (contador .EQ. 4) then; CFenol=2.673D-4; end if
!     If (contador .EQ. 5) then; CFenol=5.347D-4; end if
!     If (contador .EQ. 6) then; CFenol=1.069D-3; end if
!     If (contador .EQ. 7) then; CFenol=1.337D-3; end if
!     If (contador .EQ. 8) then; CFenol=1.604D-3; end if
!     If (contador .EQ. 9) then; CFenol=1.871D-2; end if
!     If (contador .EQ. 10) then; CFenol=2.272D-2; end if
!     If (contador .EQ. 11) then; CFenol=2.673D-2; end if
!     If (contador .EQ. 12) then; CFenol=3.074D-2; end if
!     If (contador .EQ. 13) then; CFenol=3.476D-2; end if
!     If (contador .EQ. 14) then; CFenol=3.877D-2; end if
!     If (contador .EQ. 15) then; CFenol=4.278D-2; end if
!     If (contador .EQ. 16) then; CFenol=4.679D-2; end if
!     If (contador .EQ. 17) then; CFenol=5.347D-2; end if
!     If (contador .EQ. 18) then; CFenol=1.069D-1; end if
      contador=18 ! The variable "contador" must be change from 1 to 18 to choose one of the experimental
concentrations of phenol used.
      If (contador .EQ. 1) then; CFenol=0D0; end if
      If (contador .EQ. 2) then; CFenol=5.347D-5; end if
      If (contador .EQ. 3) then; CFenol=1.337D-4; end if
      If (contador .EQ. 4) then; CFenol=2.673D-4; end if
      If (contador .EQ. 5) then; CFenol=5.347D-4; end if
      If (contador .EQ. 6) then; CFenol=1.069D-3; end if
      If (contador .EQ. 7) then; CFenol=1.337D-3; end if
      If (contador .EQ. 8) then; CFenol=1.604D-3; end if
      If (contador .EQ. 9) then; CFenol=1.871D-2; end if
      If (contador .EQ. 10) then; CFenol=2.272D-2; end if
      If (contador .EQ. 11) then; CFenol=2.673D-2; end if
      If (contador .EQ. 12) then; CFenol=3.074D-2; end if
      If (contador .EQ. 13) then; CFenol=3.476D-2; end if
      If (contador .EQ. 14) then; CFenol=3.877D-2; end if
      If (contador .EQ. 15) then; CFenol=4.278D-2; end if
      If (contador .EQ. 16) then; CFenol=4.679D-2; end if
      If (contador .EQ. 17) then; CFenol=5.347D-2; end if
      If (contador .EQ. 18) then; CFenol=1.069D-1; end if
      write(*,*) "Contador = ",contador,"CFenol = ", CFenol
!     call sleep(1)
      call MBMReGVKFR(contador,CFenol)
!     write(*,*) "CFenol = ", CFenol
!     call sleep(1)
!  End Do
   end Program BarridoFenol

   Subroutine MBMReGVKFR(contador,CFenol)
        Implicit none
   EXTERNAL  F, Jac
   DOUBLE PRECISION ATOL, RWORK, RTOL, T, Tmax,TOUT,c, Delta,Hmax,
     +Max_step,CFenol
      Integer neq,ITOL,ITASK,ISTATE,IOPT,LRW,LIW,MF,LPR,IWORK,I,contador
      DIMENSION c(51),RWORK(3100), IWORK(100)
      character*19  grafile
      Character*1 ContadorArchivo1
      Character*2 ContadorArchivo2

c   15 format(A70)
```

```
C  write(*,20)
C  20 format(2x,'Archivo para los datos de integracion: ')
C  read(*,15) grafile

   If (contador .LT. 10) then
   write(ContadorArchivo1,'(I1)') contador !convierte entero a caracter
!  ContadorArchivo=trim(ContadorArchivo) !Quita los espacios en blanco
   grafile="Ago-26-2013-"//trim(ContadorArchivo1)//".txt"
   else
   write(ContadorArchivo2,'(I2)') contador
   grafile="Ago-26-2013-"//trim(ContadorArchivo2)//".txt"
   End if

C  Aqui se especifica el número de ecuaciones diferenciales ODES
   neq=51
C *** !Estos datos son los que debe leer la subrutina, o se le deben dar

C  WRITE(*,*) 'Entre el tiempo total de la simulación: '
C  READ(*,*) Tmax
C  WRITE(*,*) 'Entre el intervalo del tiempo: '
C  READ(*,*) Delta

   Tmax=2D6 !4D4 !4D5 !2D6
   Delta=40D0

C      Concentraciones Iniciales de cada especie
C  WRITE(*,*) 'Entre la concentración inicial 1: '
C  read(*,*) c(1)
C  WRITE(*,*) 'Entre la concentración inicial 2: '
C  read(*,*) c(2)

      c(1)=0.00000001           ! c(1)= Br-
   c(2)=0.0                    ! c(2)= HOBr
   c(3)=1.29                   ! c(3)= H+
   c(4)=0.0                    ! c(4)= Br2
   c(5)=0.0                    ! c(5)= HBrO2
   c(6)=2.8896D-2              ! Concentraci≤n Ariel !MBM articulo concentraci≤n 0.15D0 ! c(6)= BrO3-
   c(7)=0.0                    ! c(7)= H2BrO2+
   c(8)=0.0                    ! c(8)= Br2O4
   c(9)=0.0                    ! c(9)= BrO2*
   c(10)=0.0                   ! c(10)= Ce+3
   c(11)=5.606D-4             ! Concentraci≤n Ariel !MBM articulo concentracion 3.56D-4          ! c(11)=
Ce+4
   c(12)=0.0                   ! c(12)= O2
   c(13)=0.0                   ! c(13)= BrMA
   c(14)=0.0                   ! c(14)= BrMA*
   c(15)=0.0                   ! c(15)= BrEETRA
   c(16)=0.0                   ! c(16)= CO2
   c(17)=0.0                   ! c(17)= BrMA(enol)
   c(18)=0.0                   ! c(18)= Br2MA
   c(19)=0.0                   ! c(19)= BrMABrO2
   c(20)=0.0                   ! c(20)= OA
   c(21)=0.0                   ! c(21)= BrTA
   c(22)=0.0                   ! c(22)= MOA
   c(23)=0.0                   ! c(23)= COOH*
   c(24)=0.0                   ! c(24)= MA*
   c(25)=2.6056D-2             ! Concentraci≤n Ariel !MBM articulo concentraci≤n 5D-2             ! c(25)=
MA
   c(26)=0.0                    ! c(26)= ETA
   c(27)=0.0                    ! c(27)= MA(enol)
   c(28)=0.0                    ! c(28)= MABrO2
      c(29)=0.0                    ! c(29)= TA
      c(30)=0.0                    ! c(30)= EETA
      c(31)=0.0                    ! c(31)= TA*
      c(32)=0.0                    ! c(32)= EEHTRA
      c(33)=0.0                    ! c(33)= TA(enol)
      c(34)=0.0                    ! c(34)= TABrO2
```

```
! *** Desde aquí se suman las reacciones del mecanismo GVKFR ***
      c(35)=CFenol              ! GVKFR: c(7)= R                    Fenol          H2BrO2+
      c(36)=0.0D0               ! GVKFR: c(8)= R*           Fenol* = H-*Fenol=0    Br2O4
      c(37)=0.0D0               ! GVKFR: c(10)= Rox1                HO-HFenol=0    Ce+3
      c(38)=0.0D0               ! GVKFR: c(11)= RBr                 Br*            Ce+4
      c(39)=0.0D0               ! GVKFR: c(12)= RBr2 O=Fenol*-OH = *O-Fenol-OH     O2
      c(40)=0.0D0               ! GVKFR: c(13)= RBr*                O=Fenol=0      BrMA
      c(41)=0.0D0               ! GVKFR: c(14)= R(BrOH)             HO-Fenol-OH    BrMA*
      c(42)=0.0D0               ! GVKFR: c(15)= Rox2                HO-Fenol-Br    BrEETRA
      c(43)=0.0D0               ! GVKFR: c(16)= OQN                 (Fenol-O)2     CO2
      c(44)=0.0D0               ! GVKFR: c(17)= RBr2*                                     BrMA(enol)
      c(45)=0.0D0               ! GVKFR: c(18)= RBr(BrOH)                             Br2MA
      c(46)=0.0D0               ! GVKFR: c(19)= Rox3                                  BrMABrO2
      c(47)=0.0D0               ! GVKFR: c(20)= BrOQN                                     OA
      c(48)=0.0D0               ! GVKFR: c(21)= Rox4                                  BrTA
! *** Las especies de las intereacciones
      c(49)=0.0D0               ! Ariel: p-Hidroxyphenol
      c(50)=0.0D0               ! Ariel: p-Hidroxyphenol radical
      c(51)=0.0D0               ! Ariel: p-Quinone

!      write(*,*) "cs = ", (c(I), I=1, neq)

   open(9, file = grafile,   status='replace')


      Hmax=     10D0            ! Maximun step size
      Max_step= 10000000D0

      TOUT = 0D0
      ITOL = 1
      RTOL = 1.D-10   !tol    : Error tolerance
      ATOL = 1.D-10   !wlimit : Error control limit
      ITASK = 1
      ISTATE = 1
      IOPT = 1
      LRW = 3100
      LIW = 100
         MF = 22 !, por LSODE suministra el jacobiano full
C     Input options
      Do lpr=5,10
          rwork(lpr)=0
     iwork(lpr)=0
      End Do
C      rwork(5)=H !h0
      rwork(6)=Hmax
      iwork(6)=Max_step
      Do While (Tout .LE. Tmax) !Del ciclo de integracion DLSODE
       CALL DLSODE(F,NEQ,c,T,TOUT,ITOL,RTOL,ATOL,ITASK,ISTATE,
     +     IOPT,RWORK,LRW,IWORK,LIW,Jac,MF)
!       WRITE(6,32) T,c(1),c(2)
!  32   FORMAT(7H En T =,E12.4,7H c(1) =,E14.6,7H c(2) =,E14.6)

       WRITE(9,34) T,((c(I)), I=1, neq)
  34   FORMAT(52(1PE13.5))
       IF (ISTATE .LT. 0) Then
      WRITE(6,90)ISTATE
  90    FORMAT(///22H ERROR HALT.. ISTATE =,I3)
      close(9)
      STOP
       End If

       TOUT = TOUT + delta
      End Do !fin Del ciclo de integracion DLSODE *********************************
      WRITE(6,60)IWORK(11),IWORK(12),IWORK(13)
  60  FORMAT(/12H NO. STEPS =,I4,11H  NO. F-S =,I4,11H  NO. J-S =,I4)
      close(9)
```

```
      END Subroutine MBMReGVKFR
C************************************************
   SUBROUTINE F (NEQ, T, c, YDOT)
      DOUBLE PRECISION T, c, YDOT, V
      DIMENSION c(51), YDOT(51), V(72)
   Integer especie
! *** Para controlar los posibles valores negativos de GVKFR R=c(7) y R*=c(8)
   Do especie=1, 51, 1
      IF ( c(especie) < 1D-14 ) Then
        c(especie)=0D0 !; write (*,*), 'Se acabo el reactivo. c(7) =', c(7)
      END IF
   End Do
      V(1)= 8d+9*c(1)*c(2)*c(3)-110*c(4)
      V(2)= 2.9d+6*c(1)*c(5)*c(3)-2d-5*c(2)*c(2)
      V(3)= 0.6*c(1)*c(6)*c(3)*c(3)-3.2*c(2)*c(5)
      V(4)= 2d+6*c(5)*c(3)-1.0d+8*c(7)
      V(5)= 1.7d+5*c(5)*c(7)
      V(6)= 48*c(5)*c(6)*c(3)-3.2d+3*c(8)
      V(7)= 7.5d+4*c(8)-1.4d+9*c(9)*c(9)
      V(8)= 6d+4*c(10)*c(9)*c(3)-1.3d+4*c(11)*c(5)
      V(9)= 6d-10*c(6)*c(6)*c(3)*c(3)
      V(10)= 0.06*c(9)
      V(11)= 0.1*c(13)*c(11)-400*c(14)*c(10)*c(3)
      V(12)= 1d+9*c(14)*c(14)
      V(13)= 1.2d-2*c(13)-800*c(17)
      V(14)= 3.5d+6*c(17)*c(4)
      V(15)= 1.1d+6*c(17)*c(2)
      V(16)= 2d+9*c(14)*c(9)
      V(17)= 0.62*c(19)
      V(18)= 0.46*c(19)
      V(19)= 1.5*c(21)
      V(20)= 7d+3*c(22)*c(11)
      V(21)= 28*c(20)*c(11)
      V(22)= 5d+9*c(23)*c(23)
      V(23)= 1d+7*c(23)*c(11)
      V(24)= 0.6d+6*c(23)*c(13)
      V(25)= 3d+9*c(23)*c(14)
      V(26)= 5d+9*c(23)*c(9)
      V(27)= 0.23*c(25)*c(11)-2.2d+4*c(24)*c(10)*c(3)
      V(28)= 3.2d+9*c(24)*c(24)
      V(29)= 2.6d-3*c(25)-180*c(27)
      V(30)= 2d+6*c(27)*c(4)
      V(31)= 6.7d+5*c(27)*c(2)
      V(32)= 5d+9*c(24)*c(9)
      V(33)= 0.55*c(28)
      V(34)= 1.0*c(28)
      V(35)= 2d+9*c(24)*c(14)
      V(36)= 4d+9*c(24)*c(23)
      V(37)= 0.66*c(29)*c(11)-1.7d+4*c(31)*c(10)*c(3)
      V(38)= 1d+9*c(31)*c(31)
      V(39)= 2.3d-5*c(29)-1.5*c(33)
      V(40)= 3d+5*c(33)*c(4)
      V(41)= 2d+5*c(33)*c(2)
      V(42)= 1d+9*c(31)*c(24)
      V(43)= 1d+9*c(31)*c(14)
      V(44)= 3d+9*c(31)*c(23)
      V(45)= 2d+9*c(31)*c(9)
      V(46)= 0.1*c(34)
      V(47)= 5d-5*c(29)*c(6)
      V(48)= 160*c(24)*c(6)*c(3)
! *** Desde aquí se suman las velocidades del mecanismo GVKFR ***
      V(49)= 1D-2*c(35)*c(6)*c(3)
      V(50)= 1D4*c(35)*c(9)
      V(51)= 1D4*c(36)*c(6)*c(3)
      V(52)= 6D5*c(35)*c(4)
      V(53)= 2D3*c(38)*c(4)
      V(54)= 1D4*c(38)*c(9)
```

```
      V(55)= 1D4*c(40)*c(6)*c(3)
      V(56)= 1D4*c(40)*c(6)*c(3)
      V(57)= 2D-1*c(41)
      V(58)= 1D4*c(39)*c(9)
      V(59)= 1D4*c(44)*c(6)*c(3)
      V(60)= 2.5D3*c(44)*c(6)*c(3)
      V(61)= 2D-1*c(45)
      V(62)= 1D2*c(4)*c(37)
! *** Estas reacciones se adiciona, digamos inter mecanismos ***
! Se usa el formato largo, propuesto por Ariel - Ver cuaderno.
      V(63)= 1D3*c(11)*c(35)-4D4*c(10)*c(36)*c(3)
      V(64)= 1D4*c(36)*c(11)
      V(65)= 1D3*c(49)*c(11)-4D4*c(50)*c(10)*c(3)
      V(66)= 1D4*c(50)*c(11)
      V(67)= 1D6*c(51)*c(23)
      V(68)= 1D8*c(50)*c(23)
      V(69)= 1D6*c(51)*c(31)
      V(70)= 1D7*c(50)*c(31)
! *** NR5 Revisado en la web http://www.phy.bme.hu/deps/chem_ph/Research/BZ_Simulation/Ce4+.html
      V(71)= 1D0*c(28)
! *** Se adiciona una reacci≤n de consumo de la quinona, para representar la desaparici≤n de esta y por
tanto la desaparici≤n del ciclo que origina los “burst”.
      V(72)= 1D5*c(51)


C          Br-
      YDOT(1)=-V(1)-V(2)-V(3)+V(12)+V(14)+V(17)+V(19)+V(24)+V(30)
     ++V(35)+V(40)+V(52)+V(53)+V(57)+V(62)+V(71)

C          HOBr
      YDOT(2)=-V(1)+2*V(2)+V(3)+V(5)-V(15)+V(17)-V(31)+V(33)-V(41)

C          H+
      YDOT(3)=-V(1)-V(2)-2*V(3)-V(4)+2*V(5)-V(6)-V(8)-2*V(9)+V(11)
     ++V(12)+V(14)+V(17)+V(19)+V(20)+V(21)+V(23)+V(24)+V(27)+V(30)
     ++V(35)+V(37)+V(40)-V(48)-V(49)-V(51)+V(52)+V(53)-V(55)-V(56)
     ++V(57)-V(59)-V(60)+V(61)+V(62)+V(63)+V(64)+V(65)+V(66)+V(71)

c          Br2
      YDOT(4)=V(1)+0.5*V(10)-V(14)-V(30)-V(40)-V(52)-V(53)-V(62)

c          HBrO2
      YDOT(5)=-V(2)+V(3)-V(4)-V(5)-V(6)+V(8)+2*V(9)+V(18)+V(26)+V(34)
     ++V(46)+V(47)+V(50)+V(54)+V(58)

c         BrO3-
      YDOT(6)=-V(3)+V(5)-V(6)-2*V(9)-V(47)-V(48)-V(49)-V(51)-V(55)
     +-V(56)-V(59)-V(60)

c          H2BrO2+
      YDOT(7)=V(4)-V(5)

c          Br2O4
      YDOT(8)=V(6)-V(7)

c          BrO2*
      YDOT(9)=2*V(7)-V(8)-V(10)-V(16)-V(26)-V(32)-V(45)+V(48)+V(49)
     +-V(50)-V(54)+V(55)+V(56)-V(58)+V(59)+V(60)

c          Ce+3
      YDOT(10)=-V(8)+V(11)+V(20)+V(21)+V(23)+V(27)+V(37)+V(63)+V(64)
     ++V(65)+V(66)

c          Ce+4
      YDOT(11)=+V(8)-V(11)-V(20)-V(21)-V(23)-V(27)-V(37)-V(63)-V(64)
     +-V(65)-V(66)
```

```
c         O2
      YDOT(12)=V(9)+V(10)      !!!!Solo se acumula

c         BrMA
      YDOT(13)=-V(11)-V(13)-V(24)+V(25)+V(30)+V(31)

c         BrMA*
      YDOT(14)=V(11)-2*V(12)-V(16)-V(25)-V(35)-V(43)

c         BrEETRA
      YDOT(15)=V(12)+V(43)     !!!!Solo se acumula

c         CO2
      YDOT(16)=V(12)+V(17)+V(21)+V(23)+V(24)+V(25)+V(26)+V(36)+V(38)
     ++V(43)+V(44)+V(67)+V(68)+V(71)             !!!!Solo se acumula

c         BrMA(enol)
      YDOT(17)=V(13)-V(14)-V(15)

c         Br2MA
      YDOT(18)=V(14)+V(15)     !!!!Solo se acumula

c         BrMABrO2
      YDOT(19)=V(16)-V(17)-V(18)

c         OA
      YDOT(20)=V(17)+V(20)-V(21)+V(22)+V(71)

c         BrTA
      YDOT(21)=V(18)-V(19)+V(40)+V(41)

c         MOA
      YDOT(22)=V(19)-V(20)+V(33)+V(46)+V(47)+V(69)+V(70)

c         COOH*
      YDOT(23)=V(20)+V(21)-2D0*V(22)-V(23)-V(24)-V(25)-V(26)-V(36)
     +-V(44)-V(67)-V(68)

c         MA*
      YDOT(24)=V(24)+V(27)-2*V(28)-V(32)-V(35)-V(36)-V(42)-V(48)

c         MA
      YDOT(25)=-V(27)-V(29)+V(36)

c         ETA
      YDOT(26)=V(28)  !!!Solo se acumula

c         MA(enol)
      YDOT(27)=V(29)-V(30)-V(31)

c         MABrO2
      YDOT(28)=V(32)-V(33)-V(34)-V(71)

c         TA
      YDOT(29)=V(34)-V(37)-V(39)+V(44)-V(47)

c         EETA
      YDOT(30)=V(35)+V(42)    !!!!Solo se acumula

c         TA*
      YDOT(31)=V(37)-2*V(38)-V(42)-V(43)-V(44)-V(45)-V(69)-V(70)

c         EEHTRA
      YDOT(32)=V(38)          !!!!Solo se acumula

c         TA(enol)
      YDOT(33)=V(39)-V(40)-V(41)
```

```
c         TABrO2
      YDOT(34)=V(45)-V(46)
! *** Desde aquφ se suman las EDOs de las especies del mecanismo GVKFR
!         R                   Fenol                          H2BrO2+
      YDOT(35)=-V(49)-V(50)-V(52)-V(63)

!         R*                  Fenol* = H-*Fenol=0            Br2O4
      YDOT(36)=V(49)+V(50)-V(51)+V(63)-V(64)

!         Rox1                HO-HFenol=0                    Ce+3
      YDOT(37)=V(51)-V(62)

!         RBr                 Br*                            Ce+4
      YDOT(38)=V(52)-V(53)-v(54)

!         RBr2                O=Fenol*-OH = *O-Fenol-OH      O2
      YDOT(39)=V(53)-V(58)

!         RBr*                O=Fenol=0                      BrMA
      YDOT(40)=V(54)-V(55)-V(56)

!         R(BrOH)             HO-Fenol-OH                    BrMA*
      YDOT(41)=V(55)-V(57)

!         Rox2                HO-Fenol-Br                    BrEETRA
      YDOT(42)=V(56)              !!!!Solo se acumula

!         OQN                 (Fenol-O)2                     CO2
      YDOT(43)=V(57)            !!!!Solo se acumula

!         RBr2*
      YDOT(44)=V(58)-V(59)-V(60)

!         RBr(BrOH)
      YDOT(45)=V(59)-V(61)

!         Rox3
      YDOT(46)=V(60)            !!!!Solo se acumula

!         BrOQN
      YDOT(47)=V(61)            !!!!Solo se acumula

!         Rox4
      YDOT(48)=V(62)            !!!!Solo se acumula
!         ROH - p-Hidroxifenol - Especie nueva
      YDOT(49)=V(64)-V(65)+V(68)+V(70)
!         p-Hidroxifenol Radical - Especie nueva
      YDOT(50)=V(65)-V(66)+V(67)-V(68)+V(69)-V(70)
!         p-Quinona - Especie nueva
      YDOT(51)=V(66)-V(67)-V(69)-V(72)
   END SUBROUTINE !F

   SUBROUTINE JAC
   END SUBROUTINE JAC
C***********************************************************************
*DECK DLSODE
      SUBROUTINE DLSODE (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,
     1                   ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF)
C***BEGIN PROLOGUE  DLSODE
C***PURPOSE  Livermore solver for ordinary differential equations.
C            DLSODE solves the initial-value problem for stiff or
C            nonstiff systems of first-order ODE's,
C               dy/dt = f(t,y),   or, in component form,
C               dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(N)),  i=1,...,N.
C***LIBRARY   MATHLIB (ODEPACK)
C***CATEGORY  I1A
```

```
C***TYPE      DOUBLE PRECISION (SLSODE-S, DLSODE-D)
C***KEYWORDS  ORDINARY DIFFERENTIAL EQUATIONS, INITIAL VALUE PROBLEM,
C             STIFF, NONSTIFF
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C             Computing and Mathematics Research Div., L-316
C             Lawrence Livermore National Laboratory
C             Livermore, CA 94550.
C***DESCRIPTION
C
C     NOTE: The DLSODE solver is not re-entrant, and so is usable on
C           the Cray multi-processor machines only if it is not used
C           in a multi-tasking environment.
C           If re-entrancy is required, use NLSODE instead.
C
C           The formats of the DLSODE and NLSODE writeups differ from
C           those of the other MATHLIB routines.
C
C           The "Usage" and "Arguments" sections treat only a subset of
C           available options, in condensed fashion.  The options
C           covered and the information supplied will support most
C           standard uses of DLSODE.
C
C           For more sophisticated uses, full details on all options are
C           given in the concluding section, headed "Long Description."
C           A synopsis of the DLSODE Long Description is provided at the
C           beginning of that section; general topics covered are:
C           - Elements of the call sequence; optional input and output
C           - Optional supplemental routines in the DLSODE package
C           - internal COMMON block
C
C *Usage:
C     Communication between the user and the DLSODE package, for normal
C     situations, is summarized here.  This summary describes a subset
C     of the available options.  See "Long Description" for complete
C     details, including optional communication, nonstandard options,
C     and instructions for special situations.
C
C     A sample program is given in the "Examples" section.
C
C     Refer to the argument descriptions for the definitions of the
C     quantities that appear in the following sample declarations.
C
C     For MF = 10,
C        PARAMETER  (LRW = 20 + 16*NEQ,           LIW = 20)
C     For MF = 21 or 22,
C        PARAMETER  (LRW = 22 +  9*NEQ + NEQ**2,  LIW = 20 + NEQ)
C     For MF = 24 or 25,
C        PARAMETER  (LRW = 22 + 10*NEQ + (2*ML+MU)*NEQ,
C       *                                LIW = 20 + NEQ)
C
C        EXTERNAL F, JAC
C        INTEGER  NEQ, ITOL, ITASK, ISTATE, IOPT, LRW, IWORK(LIW),
C       *         LIW, MF
C        DOUBLE PRECISION Y(NEQ), T, TOUT, RTOL, ATOL(ntol), RWORK(LRW)
C
C        CALL DLSODE (F, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,
C       *            ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JAC, MF)
C
C *Arguments:
C     F     :EXT    Name of subroutine for right-hand-side vector f.
C                   This name must be declared EXTERNAL in calling
C                   program.  The form of F must be:
C
C                   SUBROUTINE  F (NEQ, T, Y, YDOT)
C                   INTEGER  NEQ
C                   DOUBLE PRECISION  T, Y(NEQ), YDOT(NEQ)
C
```

```
C                  The inputs are NEQ, T, Y.  F is to set
C
C                  YDOT(i) = f(i,T,Y(1),Y(2),...,Y(NEQ)),
C                                                 i = 1, ..., NEQ .
C
C     NEQ   :IN    Number of first-order ODE's.
C
C     Y     :INOUT Array of values of the y(t) vector, of length NEQ.
C                  Input:  For the first call, Y should contain the
C                          values of y(t) at t = T. (Y is an input
C                          variable only if ISTATE = 1.)
C                  Output: On return, Y will contain the values at the
C                          new t-value.
C
C     T     :INOUT Value of the independent variable.  On return it
C                  will be the current value of t (normally TOUT).
C
C     TOUT  :IN    Next point where output is desired (.NE. T).
C
C     ITOL  :IN    1 or 2 according as ATOL (below) is a scalar or
C                  an array.
C
C     RTOL  :IN    Relative tolerance parameter (scalar).
C
C     ATOL  :IN    Absolute tolerance parameter (scalar or array).
C                  If ITOL = 1, ATOL need not be dimensioned.
C                  If ITOL = 2, ATOL must be dimensioned at least NEQ.
C
C                  The estimated local error in Y(i) will be controlled
C                  so as to be roughly less (in magnitude) than
C
C                  EWT(i) = RTOL*ABS(Y(i)) + ATOL     if ITOL = 1, or
C                  EWT(i) = RTOL*ABS(Y(i)) + ATOL(i)  if ITOL = 2.
C
C                  Thus the local error test passes if, in each
C                  component, either the absolute error is less than
C                  ATOL (or ATOL(i)), or the relative error is less
C                  than RTOL.
C
C                  Use RTOL = 0.0 for pure absolute error control, and
C                  use ATOL = 0.0 (or ATOL(i) = 0.0) for pure relative
C                  error control.  Caution:  Actual (global) errors may
C                  exceed these local tolerances, so choose them
C                  conservatively.
C
C     ITASK :IN    Flag indicating the task DLSODE is to perform.
C                  Use ITASK = 1 for normal computation of output
C                  values of y at t = TOUT.
C
C     ISTATE:INOUT Index used for input and output to specify the state
C                  of the calculation.
C                  Input:
C                   1   This is the first call for a problem.
C                   2   This is a subsequent call.
C                  Output:
C                   2   DLSODE was successful (otherwise, negative).
C                       Note that ISTATE need not be modified after a
C                       successful return.
C                  -1   Excess work done on this call (perhaps wrong
C                       MF).
C                  -2   Excess accuracy requested (tolerances too
C                       small).
C                  -3   Illegal input detected (see printed message).
C                  -4   Repeated error test failures (check all
C                       inputs).
C                  -5   Repeated convergence failures (perhaps bad
C                       Jacobian supplied or wrong choice of MF or
```

```
C                     tolerances).
C                 -6  Error weight became zero during problem
C                     (solution component i vanished, and ATOL or
C                     ATOL(i) = 0.).
C
C     IOPT  :IN     Flag indicating whether optional inputs are used:
C                   0   No.
C                   1   Yes.  (See "Optional inputs" under "Long
C                       Description," Part 1.)
C
C     RWORK :WORK   Real work array of length at least:
C                   20 + 16*NEQ                    for MF = 10,
C                   22 +  9*NEQ + NEQ**2           for MF = 21 or 22,
C                   22 + 10*NEQ + (2*ML + MU)*NEQ  for MF = 24 or 25.
C
C     LRW   :IN     Declared length of RWORK (in user's DIMENSION
C                   statement).
C
C     IWORK :WORK   Integer work array of length at least:
C                   20        for MF = 10,
C                   20 + NEQ  for MF = 21, 22, 24, or 25.
C
C                   If MF = 24 or 25, input in IWORK(1),IWORK(2) the
C                   lower and upper Jacobian half-bandwidths ML,MU.
C
C                   On return, IWORK contains information that may be
C                   of interest to the user:
C
C            Name   Location   Meaning
C            -----  ---------  -----------------------------------------
C            NST    IWORK(11)  Number of steps taken for the problem so
C                              far.
C            NFE    IWORK(12)  Number of f evaluations for the problem
C                              so far.
C            NJE    IWORK(13)  Number of Jacobian evaluations (and of
C                              matrix LU decompositions) for the problem
C                              so far.
C            NQU    IWORK(14)  Method order last used (successfully).
C            LENRW  IWORK(17)  Length of RWORK actually required.  This
C                              is defined on normal returns and on an
C                              illegal input return for insufficient
C                              storage.
C            LENIW  IWORK(18)  Length of IWORK actually required.  This
C                              is defined on normal returns and on an
C                              illegal input return for insufficient
C                              storage.
C
C     LIW   :IN     Declared length of IWORK (in user's DIMENSION
C                   statement).
C
C     JAC   :EXT    Name of subroutine for Jacobian matrix (MF =
C                   21 or 24).  If used, this name must be declared
C                   EXTERNAL in calling program.  If not used, pass a
C                   dummy name.  The form of JAC must be:
C
C                   SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
C                   INTEGER  NEQ, ML, MU, NROWPD
C                   DOUBLE PRECISION  T, Y(NEQ), PD(NROWPD,NEQ)
C
C                   See item c, under "Description" below for more
C                   information about JAC.
C
C     MF    :IN     Method flag.  Standard values are:
C                   10  Nonstiff (Adams) method, no Jacobian used.
C                   21  Stiff (BDF) method, user-supplied full Jacobian.
C                   22  Stiff method, internally generated full
C                       Jacobian.
```

```
C                  24  Stiff method, user-supplied banded Jacobian.
C                  25  Stiff method, internally generated banded
C                      Jacobian.
C
C *Description:
C     DLSODE solves the initial value problem for stiff or nonstiff
C     systems of first-order ODE's,
C
C        dy/dt = f(t,y) ,
C
C     or, in component form,
C
C        dy(i)/dt = f(i) = f(i,t,y(1),y(2),...,y(NEQ))
C                                                (i = 1, ..., NEQ) .
C
C     DLSODE is a package based on the GEAR and GEARB packages, and on
C     the October 23, 1978, version of the tentative ODEPACK user
C     interface standard, with minor modifications.
C
C     The steps in solving such a problem are as follows.
C
C     a. First write a subroutine of the form
C
C           SUBROUTINE  F (NEQ, T, Y, YDOT)
C           INTEGER  NEQ
C           DOUBLE PRECISION  T, Y(NEQ), YDOT(NEQ)
C
C        which supplies the vector function f by loading YDOT(i) with
C        f(i).
C
C     b. Next determine (or guess) whether or not the problem is stiff.
C        Stiffness occurs when the Jacobian matrix df/dy has an
C        eigenvalue whose real part is negative and large in magnitude
C        compared to the reciprocal of the t span of interest.  If the
C        problem is nonstiff, use method flag MF = 10.  If it is stiff,
C        there are four standard choices for MF, and DLSODE requires the
C        Jacobian matrix in some form.  This matrix is regarded either
C        as full (MF = 21 or 22), or banded (MF = 24 or 25).  In the
C        banded case, DLSODE requires two half-bandwidth parameters ML
C        and MU. These are, respectively, the widths of the lower and
C        upper parts of the band, excluding the main diagonal.  Thus the
C        band consists of the locations (i,j) with
C
C           i - ML <= j <= i + MU ,
C
C        and the full bandwidth is ML + MU + 1 .
C
C     c. If the problem is stiff, you are encouraged to supply the
C        Jacobian directly (MF = 21 or 24), but if this is not feasible,
C        DLSODE will compute it internally by difference quotients (MF =
C        22 or 25).  If you are supplying the Jacobian, write a
C        subroutine of the form
C
C           SUBROUTINE  JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
C           INTEGER  NEQ, ML, MU, NRWOPD
C           DOUBLE PRECISION  Y, Y(NEQ), PD(NROWPD,NEQ)
C
C        which provides df/dy by loading PD as follows:
C        - For a full Jacobian (MF = 21), load PD(i,j) with df(i)/dy(j),
C          the partial derivative of f(i) with respect to y(j).  (Ignore
C          the ML and MU arguments in this case.)
C        - For a banded Jacobian (MF = 24), load PD(i-j+MU+1,j) with
C          df(i)/dy(j); i.e., load the diagonal lines of df/dy into the
C          rows of PD from the top down.
C        - In either case, only nonzero elements need be loaded.
C
C     d. Write a main program that calls subroutine DLSODE once for each
```

```
C        point at which answers are desired.  This should also provide
C        for possible use of logical unit 6 for output of error messages
C        by DLSODE.
C
C        Before the first call to DLSODE, set ISTATE = 1, set Y and T to
C        the initial values, and set TOUT to the first output point.  To
C        continue the integration after a successful return, simply
C        reset TOUT and call DLSODE again.  No other parameters need be
C        reset.
C
C *Examples:
C     The following is a simple example problem, with the coding needed
C     for its solution by DLSODE. The problem is from chemical kinetics,
C     and consists of the following three rate equations:
C
C        dy1/dt = -.04*y1 + 1.E4*y2*y3
C        dy2/dt = .04*y1 - 1.E4*y2*y3 - 3.E7*y2**2
C        dy3/dt = 3.E7*y2**2
C
C     on the interval from t = 0.0 to t = 4.E10, with initial conditions
C     y1 = 1.0, y2 = y3 = 0. The problem is stiff.
C
C     The following coding solves this problem with DLSODE, using
C     MF = 21 and printing results at t = .4, 4., ..., 4.E10.  It uses
C     ITOL = 2 and ATOL much smaller for y2 than for y1 or y3 because y2
C     has much smaller values.  At the end of the run, statistical
C     quantities of interest are printed.
C
C        EXTERNAL  FEX, JEX
C        INTEGER  IOPT, IOUT, ISTATE, ITASK, ITOL, IWORK(23), LIW, LRW,
C       *         MF, NEQ
C        DOUBLE PRECISION  ATOL(3), RTOL, RWORK(58), T, TOUT, Y(3)
C        NEQ = 3
C        Y(1) = 1.D0
C        Y(2) = 0.D0
C        Y(3) = 0.D0
C        T = 0.D0
C        TOUT = .4D0
C        ITOL = 2
C        RTOL = 1.D-4
C        ATOL(1) = 1.D-6
C        ATOL(2) = 1.D-10
C        ATOL(3) = 1.D-6
C        ITASK = 1
C        ISTATE = 1
C        IOPT = 0
C        LRW = 58
C        LIW = 23
C        MF = 21
C        DO 40 IOUT = 1,12
C          CALL DLSODE (FEX, NEQ, Y, T, TOUT, ITOL, RTOL, ATOL, ITASK,
C       *               ISTATE, IOPT, RWORK, LRW, IWORK, LIW, JEX, MF)
C          WRITE(6,20)  T, Y(1), Y(2), Y(3)
C    20    FORMAT(' At t =',D12.4,'   y =',3D14.6)
C          IF (ISTATE .LT. 0)  GO TO 80
C    40    TOUT = TOUT*10.D0
C        WRITE(6,60)  IWORK(11), IWORK(12), IWORK(13)
C    60  FORMAT(/' No. steps =',i4,',  No. f-s =',i4,',  No. J-s =',i4)
C        STOP
C    80  WRITE(6,90)  ISTATE
C    90  FORMAT(///' Error halt.. ISTATE =',I3)
C        STOP
C        END
C
C        SUBROUTINE  FEX (NEQ, T, Y, YDOT)
C        INTEGER  NEQ
C        DOUBLE PRECISION  T, Y(3), YDOT(3)
```

```
C        YDOT(1) = -.04D0*Y(1) + 1.D4*Y(2)*Y(3)
C        YDOT(3) = 3.D7*Y(2)*Y(2)
C        YDOT(2) = -YDOT(1) - YDOT(3)
C        RETURN
C        END
C
C        SUBROUTINE  JEX (NEQ, T, Y, ML, MU, PD, NRPD)
C        INTEGER  NEQ, ML, MU, NRPD
C        DOUBLE PRECISION  T, Y(3), PD(NRPD,3)
C        PD(1,1) = -.04D0
C        PD(1,2) = 1.D4*Y(3)
C        PD(1,3) = 1.D4*Y(2)
C        PD(2,1) = .04D0
C        PD(2,3) = -PD(1,3)
C        PD(3,2) = 6.D7*Y(2)
C        PD(2,2) = -PD(1,2) - PD(3,2)
C        RETURN
C        END
C
C     The output from this program (on a Cray-1 in single precision)
C     is as follows.
C
C     At t =  4.0000e-01   y =  9.851726e-01  3.386406e-05  1.479357e-02
C     At t =  4.0000e+00   y =  9.055142e-01  2.240418e-05  9.446344e-02
C     At t =  4.0000e+01   y =  7.158050e-01  9.184616e-06  2.841858e-01
C     At t =  4.0000e+02   y =  4.504846e-01  3.222434e-06  5.495122e-01
C     At t =  4.0000e+03   y =  1.831701e-01  8.940379e-07  8.168290e-01
C     At t =  4.0000e+04   y =  3.897016e-02  1.621193e-07  9.610297e-01
C     At t =  4.0000e+05   y =  4.935213e-03  1.983756e-08  9.950648e-01
C     At t =  4.0000e+06   y =  5.159269e-04  2.064759e-09  9.994841e-01
C     At t =  4.0000e+07   y =  5.306413e-05  2.122677e-10  9.999469e-01
C     At t =  4.0000e+08   y =  5.494530e-06  2.197825e-11  9.999945e-01
C     At t =  4.0000e+09   y =  5.129458e-07  2.051784e-12  9.999995e-01
C     At t =  4.0000e+10   y = -7.170603e-08 -2.868241e-13  1.000000e+00
C
C     No. steps = 330,  No. f-s = 405,  No. J-s = 69
C
C *Accuracy:
C     The accuracy of the solution depends on the choice of tolerances
C     RTOL and ATOL.  Actual (global) errors may exceed these local
C     tolerances, so choose them conservatively.
C
C *Cautions:
C     The work arrays should not be altered between calls to DLSODE for
C     the same problem, except possibly for the conditional and optional
C     inputs.
C
C *Portability:
C     Since NEQ is dimensioned inside DLSODE, some compilers may object
C     to a call to DLSODE with NEQ a scalar variable.  In this event,
C     use DIMENSION NEQ(1).  Similar remarks apply to RTOL and ATOL.
C
C     Note to Cray users:
C     For maximum efficiency, use the CFT77 compiler.  Appropriate
C     compiler optimization directives have been inserted for CFT77
C     (but not CIVIC).
C
C     NOTICE:  If moving the DLSODE source code to other systems,
C     contact the author for notes on nonstandard Fortran usage,
C     COMMON block, and other installation details.
C
C *Reference:
C     Alan C. Hindmarsh, "ODEPACK, a systematized collection of ODE
C     solvers," in Scientific Computing, R. S. Stepleman, et al., Eds.
C     (North-Holland, Amsterdam, 1983), pp. 55-64.
C
C *Long Description:
```

```
C     The following complete description of the user interface to
C     DLSODE consists of four parts:
C
C     1.  The call sequence to subroutine DLSODE, which is a driver
C         routine for the solver.  This includes descriptions of both
C         the call sequence arguments and user-supplied routines.
C         Following these descriptions is a description of optional
C         inputs available through the call sequence, and then a
C         description of optional outputs in the work arrays.
C
C     2.  Descriptions of other routines in the DLSODE package that may
C         be (optionally) called by the user.  These provide the ability
C         to alter error message handling, save and restore the internal
C         COMMON, and obtain specified derivatives of the solution y(t).
C
C     3.  Descriptions of COMMON block to be declared in overlay or
C         similar environments, or to be saved when doing an interrupt
C         of the problem and continued solution later.
C
C     4.  Description of two routines in the DLSODE package, either of
C         which the user may replace with his own version, if desired.
C         These relate to the measurement of errors.
C
C
C                       Part 1.  Call Sequence
C                       ----------------------
C
C     Arguments
C     ---------
C     The call sequence parameters used for input only are
C
C        F, NEQ, TOUT, ITOL, RTOL, ATOL, ITASK, IOPT, LRW, LIW, JAC, MF,
C
C     and those used for both input and output are
C
C        Y, T, ISTATE.
C
C     The work arrays RWORK and IWORK are also used for conditional and
C     optional inputs and optional outputs.  (The term output here
C     refers to the return from subroutine DLSODE to the user's calling
C     program.)
C
C     The legality of input parameters will be thoroughly checked on the
C     initial call for the problem, but not checked thereafter unless a
C     change in input parameters is flagged by ISTATE = 3 on input.
C
C     The descriptions of the call arguments are as follows.
C
C     F        The name of the user-supplied subroutine defining the ODE
C              system.  The system must be put in the first-order form
C              dy/dt = f(t,y), where f is a vector-valued function of
C              the scalar t and the vector y. Subroutine F is to compute
C              the function f. It is to have the form
C
C                 SUBROUTINE F (NEQ, T, Y, YDOT)
C                 DOUBLE PRECISION  Y(NEQ), YDOT(NEQ)
C
C              where NEQ, T, and Y are input, and the array YDOT =
C              f(T,Y) is output.  Y and YDOT are arrays of length NEQ.
C              Subroutine F should not alter Y(1),...,Y(NEQ).  F must be
C              declared EXTERNAL in the calling program.
C
C              Subroutine F may access user-defined quantities in
C              NEQ(2),... and/or in Y(NEQ(1)+1),..., if NEQ is an array
C              (dimensioned in F) and/or Y has length exceeding NEQ(1).
C              See the descriptions of NEQ and Y below.
C
```

```
C             If quantities computed in the F routine are needed
C             externally to DLSODE, an extra call to F should be made
C             for this purpose, for consistent and accurate results.
C             If only the derivative dy/dt is needed, use DINTDY
C             instead.
C
C     NEQ     The size of the ODE system (number of first-order
C             ordinary differential equations).  Used only for input.
C             NEQ may be decreased, but not increased, during the
C             problem.  If NEQ is decreased (with ISTATE = 3 on input),
C             the remaining components of Y should be left undisturbed,
C             if these are to be accessed in F and/or JAC.
C
C             Normally, NEQ is a scalar, and it is generally referred
C             to as a scalar in this user interface description.
C             However, NEQ may be an array, with NEQ(1) set to the
C             system size.  (The DLSODE package accesses only NEQ(1).)
C             In either case, this parameter is passed as the NEQ
C             argument in all calls to F and JAC.  Hence, if it is an
C             array, locations NEQ(2),... may be used to store other
C             integer data and pass it to F and/or JAC.  Subroutines
C             F and/or JAC must include NEQ in a DIMENSION statement
C             in that case.
C
C     Y       A real array for the vector of dependent variables, of
C             length NEQ or more.  Used for both input and output on
C             the first call (ISTATE = 1), and only for output on
C             other calls.  On the first call, Y must contain the
C             vector of initial values.  On output, Y contains the
C             computed solution vector, evaluated at T. If desired,
C             the Y array may be used for other purposes between
C             calls to the solver.
C
C             This array is passed as the Y argument in all calls to F
C             and JAC.  Hence its length may exceed NEQ, and locations
C             Y(NEQ+1),... may be used to store other real data and
C             pass it to F and/or JAC.  (The DLSODE package accesses
C             only Y(1),...,Y(NEQ).)
C
C     T       The independent variable.  On input, T is used only on
C             the first call, as the initial point of the integration.
C             On output, after each call, T is the value at which a
C             computed solution Y is evaluated (usually the same as
C             TOUT).  On an error return, T is the farthest point
C             reached.
C
C     TOUT    The next value of T at which a computed solution is
C             desired.  Used only for input.
C
C             When starting the problem (ISTATE = 1), TOUT may be equal
C             to T for one call, then should not equal T for the next
C             call.  For the initial T, an input value of TOUT .NE. T
C             is used in order to determine the direction of the
C             integration (i.e., the algebraic sign of the step sizes)
C             and the rough scale of the problem.  Integration in
C             either direction (forward or backward in T) is permitted.
C
C             If ITASK = 2 or 5 (one-step modes), TOUT is ignored
C             after the first call (i.e., the first call with
C             TOUT .NE. T).  Otherwise, TOUT is required on every call.
C
C             If ITASK = 1, 3, or 4, the values of TOUT need not be
C             monotone, but a value of TOUT which backs up is limited
C             to the current internal T interval, whose endpoints are
C             TCUR - HU and TCUR.  (See "Optional Outputs" below for
C             TCUR and HU.)
C
```

```
C
C     ITOL     An indicator for the type of error control.  See
C              description below under ATOL.  Used only for input.
C
C     RTOL     A relative error tolerance parameter, either a scalar or
C              an array of length NEQ.  See description below under
C              ATOL.  Input only.
C
C     ATOL     An absolute error tolerance parameter, either a scalar or
C              an array of length NEQ.  Input only.
C
C              The input parameters ITOL, RTOL, and ATOL determine the
C              error control performed by the solver.  The solver will
C              control the vector e = (e(i)) of estimated local errors
C              in Y, according to an inequality of the form
C
C                 rms-norm of ( e(i)/EWT(i) ) <= 1,
C
C              where
C
C                 EWT(i) = RTOL(i)*ABS(Y(i)) + ATOL(i),
C
C              and the rms-norm (root-mean-square norm) here is
C
C                 rms-norm(v) = SQRT(sum v(i)**2 / NEQ).
C
C              Here EWT = (EWT(i)) is a vector of weights which must
C              always be positive, and the values of RTOL and ATOL
C              should all be nonnegative.  The following table gives the
C              types (scalar/array) of RTOL and ATOL, and the
C              corresponding form of EWT(i).
C
C              ITOL    RTOL      ATOL      EWT(i)
C              ----    ------    ------    -----------------------------
C              1       scalar    scalar    RTOL*ABS(Y(i)) + ATOL
C              2       scalar    array     RTOL*ABS(Y(i)) + ATOL(i)
C              3       array     scalar    RTOL(i)*ABS(Y(i)) + ATOL
C              4       array     array     RTOL(i)*ABS(Y(i)) + ATOL(i)
C
C              When either of these parameters is a scalar, it need not
C              be dimensioned in the user's calling program.
C
C              If none of the above choices (with ITOL, RTOL, and ATOL
C              fixed throughout the problem) is suitable, more general
C              error controls can be obtained by substituting
C              user-supplied routines for the setting of EWT and/or for
C              the norm calculation.  See Part 4 below.
C
C              If global errors are to be estimated by making a repeated
C              run on the same problem with smaller tolerances, then all
C              components of RTOL and ATOL (i.e., of EWT) should be
C              scaled down uniformly.
C
C     ITASK    An index specifying the task to be performed.  Input
C              only.  ITASK has the following values and meanings:
C              1   Normal computation of output values of y(t) at
C                  t = TOUT (by overshooting and interpolating).
C              2   Take one step only and return.
C              3   Stop at the first internal mesh point at or beyond
C                  t = TOUT and return.
C              4   Normal computation of output values of y(t) at
C                  t = TOUT but without overshooting t = TCRIT.  TCRIT
C                  must be input as RWORK(1).  TCRIT may be equal to or
C                  beyond TOUT, but not behind it in the direction of
C                  integration.  This option is useful if the problem
C                  has a singularity at or beyond t = TCRIT.
C              5   Take one step, without passing TCRIT, and return.
```

```
C                  TCRIT must be input as RWORK(1).
C
C              Note:  If ITASK = 4 or 5 and the solver reaches TCRIT
C              (within roundoff), it will return T = TCRIT (exactly) to
C              indicate this (unless ITASK = 4 and TOUT comes before
C              TCRIT, in which case answers at T = TOUT are returned
C              first).
C
C     ISTATE   An index used for input and output to specify the state
C              of the calculation.
C
C              On input, the values of ISTATE are as follows:
C              1   This is the first call for the problem
C                  (initializations will be done).  See "Note" below.
C              2   This is not the first call, and the calculation is to
C                  continue normally, with no change in any input
C                  parameters except possibly TOUT and ITASK.  (If ITOL,
C                  RTOL, and/or ATOL are changed between calls with
C                  ISTATE = 2, the new values will be used but not
C                  tested for legality.)
C              3   This is not the first call, and the calculation is to
C                  continue normally, but with a change in input
C                  parameters other than TOUT and ITASK.  Changes are
C                  allowed in NEQ, ITOL, RTOL, ATOL, IOPT, LRW, LIW, MF,
C                  ML, MU, and any of the optional inputs except H0.
C                  (See IWORK description for ML and MU.)
C
C              Note:  A preliminary call with TOUT = T is not counted as
C              a first call here, as no initialization or checking of
C              input is done.  (Such a call is sometimes useful for the
C              purpose of outputting the initial conditions.)  Thus the
C              first call for which TOUT .NE. T requires ISTATE = 1 on
C              input.
C
C              On output, ISTATE has the following values and meanings:
C               1  Nothing was done, as TOUT was equal to T with
C                  ISTATE = 1 on input.
C               2  The integration was performed successfully.
C              -1  An excessive amount of work (more than MXSTEP steps)
C                  was done on this call, before completing the
C                  requested task, but the integration was otherwise
C                  successful as far as T. (MXSTEP is an optional input
C                  and is normally 500.)  To continue, the user may
C                  simply reset ISTATE to a value >1 and call again (the
C                  excess work step counter will be reset to 0).  In
C                  addition, the user may increase MXSTEP to avoid this
C                  error return; see "Optional Inputs" below.
C              -2  Too much accuracy was requested for the precision of
C                  the machine being used.  This was detected before
C                  completing the requested task, but the integration
C                  was successful as far as T. To continue, the
C                  tolerance parameters must be reset, and ISTATE must
C                  be set to 3. The optional output TOLSF may be used
C                  for this purpose.  (Note:  If this condition is
C                  detected before taking any steps, then an illegal
C                  input return (ISTATE = -3) occurs instead.)
C              -3  Illegal input was detected, before taking any
C                  integration steps.  See written message for details.
C                  (Note:  If the solver detects an infinite loop of
C                  calls to the solver with illegal input, it will cause
C                  the run to stop.)
C              -4  There were repeated error-test failures on one
C                  attempted step, before completing the requested task,
C                  but the integration was successful as far as T.  The
C                  problem may have a singularity, or the input may be
C                  inappropriate.
C              -5  There were repeated convergence-test failures on one
```

```
C                  attempted step, before completing the requested task,
C                  but the integration was successful as far as T. This
C                  may be caused by an inaccurate Jacobian matrix, if
C                  one is being used.
C               -6 EWT(i) became zero for some i during the integration.
C                  Pure relative error control (ATOL(i)=0.0) was
C                  requested on a variable which has now vanished.  The
C                  integration was successful as far as T.
C
C               Note:  Since the normal output value of ISTATE is 2, it
C               does not need to be reset for normal continuation.  Also,
C               since a negative input value of ISTATE will be regarded
C               as illegal, a negative output value requires the user to
C               change it, and possibly other inputs, before calling the
C               solver again.
C
C     IOPT      An integer flag to specify whether any optional inputs
C               are being used on this call.  Input only.  The optional
C               inputs are listed under a separate heading below.
C               0   No optional inputs are being used.  Default values
C                   will be used in all cases.
C               1   One or more optional inputs are being used.
C
C     RWORK     A real working array (double precision).  The length of
C               RWORK must be at least
C
C                  20 + NYH*(MAXORD + 1) + 3*NEQ + LWM
C
C               where
C                  NYH = the initial value of NEQ,
C               MAXORD = 12 (if METH = 1) or 5 (if METH = 2) (unless a
C                        smaller value is given as an optional input),
C                  LWM = 0           if MITER = 0,
C                  LWM = NEQ**2 + 2  if MITER = 1 or 2,
C                  LWM = NEQ + 2     if MITER = 3, and
C                  LWM = (2*ML + MU + 1)*NEQ + 2
C                                    if MITER = 4 or 5.
C               (See the MF description below for METH and MITER.)
C
C               Thus if MAXORD has its default value and NEQ is constant,
C               this length is:
C               20 + 16*NEQ                    for MF = 10,
C               22 + 16*NEQ + NEQ**2           for MF = 11 or 12,
C               22 + 17*NEQ                    for MF = 13,
C               22 + 17*NEQ + (2*ML + MU)*NEQ  for MF = 14 or 15,
C               20 +  9*NEQ                    for MF = 20,
C               22 +  9*NEQ + NEQ**2           for MF = 21 or 22,
C               22 + 10*NEQ                    for MF = 23,
C               22 + 10*NEQ + (2*ML + MU)*NEQ  for MF = 24 or 25.
C
C               The first 20 words of RWORK are reserved for conditional
C               and optional inputs and optional outputs.
C
C               The following word in RWORK is a conditional input:
C               RWORK(1) = TCRIT, the critical value of t which the
C                          solver is not to overshoot.  Required if ITASK
C                          is 4 or 5, and ignored otherwise.  See ITASK.
C
C     LRW       The length of the array RWORK, as declared by the user.
C               (This will be checked by the solver.)
C
C     IWORK     An integer work array.  Its length must be at least
C               20       if MITER = 0 or 3 (MF = 10, 13, 20, 23), or
C               20 + NEQ otherwise (MF = 11, 12, 14, 15, 21, 22, 24, 25).
C               (See the MF description below for MITER.)  The first few
C               words of IWORK are used for conditional and optional
C               inputs and optional outputs.
```

```
C
C               The following two words in IWORK are conditional inputs:
C               IWORK(1) = ML   These are the lower and upper half-
C               IWORK(2) = MU   bandwidths, respectively, of the banded
C                               Jacobian, excluding the main diagonal.
C                          The band is defined by the matrix locations
C                          (i,j) with i - ML <= j <= i + MU. ML and MU
C                          must satisfy 0 <= ML,MU <= NEQ - 1. These are
C                          required if MITER is 4 or 5, and ignored
C                          otherwise.  ML and MU may in fact be the band
C                          parameters for a matrix to which df/dy is only
C                          approximately equal.
C
C     LIW       The length of the array IWORK, as declared by the user.
C               (This will be checked by the solver.)
C
C     Note:  The work arrays must not be altered between calls to DLSODE
C     for the same problem, except possibly for the conditional and
C     optional inputs, and except for the last 3*NEQ words of RWORK.
C     The latter space is used for internal scratch space, and so is
C     available for use by the user outside DLSODE between calls, if
C     desired (but not for use by F or JAC).
C
C     JAC       The name of the user-supplied routine (MITER = 1 or 4) to
C               compute the Jacobian matrix, df/dy, as a function of the
C               scalar t and the vector y.  (See the MF description below
C               for MITER.)  It is to have the form
C
C                  SUBROUTINE JAC (NEQ, T, Y, ML, MU, PD, NROWPD)
C                  DOUBLE PRECISION  Y(NEQ), PD(NROWPD,NEQ)
C
C               where NEQ, T, Y, ML, MU, and NROWPD are input and the
C               array PD is to be loaded with partial derivatives
C               (elements of the Jacobian matrix) on output.  PD must be
C               given a first dimension of NROWPD.  T and Y have the same
C               meaning as in subroutine F.
C
C               In the full matrix case (MITER = 1), ML and MU are
C               ignored, and the Jacobian is to be loaded into PD in
C               columnwise manner, with df(i)/dy(j) loaded into PD(i,j).
C
C               In the band matrix case (MITER = 4), the elements within
C               the band are to be loaded into PD in columnwise manner,
C               with diagonal lines of df/dy loaded into the rows of PD.
C               Thus df(i)/dy(j) is to be loaded into PD(i-j+MU+1,j).  ML
C               and MU are the half-bandwidth parameters (see IWORK).
C               The locations in PD in the two triangular areas which
C               correspond to nonexistent matrix elements can be ignored
C               or loaded arbitrarily, as they are overwritten by DLSODE.
C
C               JAC need not provide df/dy exactly. A crude approximation
C               (possibly with a smaller bandwidth) will do.
C
C               In either case, PD is preset to zero by the solver, so
C               that only the nonzero elements need be loaded by JAC.
C               Each call to JAC is preceded by a call to F with the same
C               arguments NEQ, T, and Y. Thus to gain some efficiency,
C               intermediate quantities shared by both calculations may
C               be saved in a user COMMON block by F and not recomputed
C               by JAC, if desired.  Also, JAC may alter the Y array, if
C               desired.  JAC must be declared EXTERNAL in the calling
C               program.
C
C               Subroutine JAC may access user-defined quantities in
C               NEQ(2),... and/or in Y(NEQ(1)+1),... if NEQ is an array
C               (dimensioned in JAC) and/or Y has length exceeding
C               NEQ(1).  See the descriptions of NEQ and Y above.
```

```
C
C     MF       The method flag.  Used only for input.  The legal values
C              of MF are 10, 11, 12, 13, 14, 15, 20, 21, 22, 23, 24,
C              and 25.  MF has decimal digits METH and MITER:
C                 MF = 10*METH + MITER .
C
C              METH indicates the basic linear multistep method:
C              1   Implicit Adams method.
C              2   Method based on backward differentiation formulas
C                  (BDF's).
C
C              MITER indicates the corrector iteration method:
C              0   Functional iteration (no Jacobian matrix is
C                  involved).
C              1   Chord iteration with a user-supplied full (NEQ by
C                  NEQ) Jacobian.
C              2   Chord iteration with an internally generated
C                  (difference quotient) full Jacobian (using NEQ
C                  extra calls to F per df/dy value).
C              3   Chord iteration with an internally generated
C                  diagonal Jacobian approximation (using one extra call
C                  to F per df/dy evaluation).
C              4   Chord iteration with a user-supplied banded Jacobian.
C              5   Chord iteration with an internally generated banded
C                  Jacobian (using ML + MU + 1 extra calls to F per
C                  df/dy evaluation).
C
C              If MITER = 1 or 4, the user must supply a subroutine JAC
C              (the name is arbitrary) as described above under JAC.
C              For other values of MITER, a dummy argument can be used.
C
C     Optional Inputs
C     ---------------
C     The following is a list of the optional inputs provided for in the
C     call sequence.  (See also Part 2.)  For each such input variable,
C     this table lists its name as used in this documentation, its
C     location in the call sequence, its meaning, and the default value.
C     The use of any of these inputs requires IOPT = 1, and in that case
C     all of these inputs are examined.  A value of zero for any of
C     these optional inputs will cause the default value to be used.
C     Thus to use a subset of the optional inputs, simply preload
C     locations 5 to 10 in RWORK and IWORK to 0.0 and 0 respectively,
C     and then set those of interest to nonzero values.
C
C     Name    Location   Meaning and default value
C     ------  ---------  -----------------------------------------------
C     H0      RWORK(5)   Step size to be attempted on the first step.
C                        The default value is determined by the solver.
C     HMAX    RWORK(6)   Maximum absolute step size allowed.  The
C                        default value is infinite.
C     HMIN    RWORK(7)   Minimum absolute step size allowed.  The
C                        default value is 0.  (This lower bound is not
C                        enforced on the final step before reaching
C                        TCRIT when ITASK = 4 or 5.)
C     MAXORD  IWORK(5)   Maximum order to be allowed.  The default value
C                        is 12 if METH = 1, and 5 if METH = 2. (See the
C                        MF description above for METH.)  If MAXORD
C                        exceeds the default value, it will be reduced
C                        to the default value.  If MAXORD is changed
C                        during the problem, it may cause the current
C                        order to be reduced.
C     MXSTEP  IWORK(6)   Maximum number of (internally defined) steps
C                        allowed during one call to the solver.  The
C                        default value is 500.
C     MXHNIL  IWORK(7)   Maximum number of messages printed (per
C                        problem) warning that T + H = T on a step
C                        (H = step size).  This must be positive to
```

```
C                       result in a nondefault value.  The default
C                       value is 10.
C
C     Optional Outputs
C     ----------------
C     As optional additional output from DLSODE, the variables listed
C     below are quantities related to the performance of DLSODE which
C     are available to the user.  These are communicated by way of the
C     work arrays, but also have internal mnemonic names as shown.
C     Except where stated otherwise, all of these outputs are defined on
C     any successful return from DLSODE, and on any return with ISTATE =
C     -1, -2, -4, -5, or -6.  On an illegal input return (ISTATE = -3),
C     they will be unchanged from their existing values (if any), except
C     possibly for TOLSF, LENRW, and LENIW.  On any error return,
C     outputs relevant to the error will be defined, as noted below.
C
C     Name   Location   Meaning
C     -----  ---------  ------------------------------------------------
C     HU     RWORK(11)  Step size in t last used (successfully).
C     HCUR   RWORK(12)  Step size to be attempted on the next step.
C     TCUR   RWORK(13)  Current value of the independent variable which
C                       the solver has actually reached, i.e., the
C                       current internal mesh point in t. On output,
C                       TCUR will always be at least as far as the
C                       argument T, but may be farther (if interpolation
C                       was done).
C     TOLSF  RWORK(14)  Tolerance scale factor, greater than 1.0,
C                       computed when a request for too much accuracy
C                       was detected (ISTATE = -3 if detected at the
C                       start of the problem, ISTATE = -2 otherwise).
C                       If ITOL is left unaltered but RTOL and ATOL are
C                       uniformly scaled up by a factor of TOLSF for the
C                       next call, then the solver is deemed likely to
C                       succeed.  (The user may also ignore TOLSF and
C                       alter the tolerance parameters in any other way
C                       appropriate.)
C     NST    IWORK(11)  Number of steps taken for the problem so far.
C     NFE    IWORK(12)  Number of F evaluations for the problem so far.
C     NJE    IWORK(13)  Number of Jacobian evaluations (and of matrix LU
C                       decompositions) for the problem so far.
C     NQU    IWORK(14)  Method order last used (successfully).
C     NQCUR  IWORK(15)  Order to be attempted on the next step.
C     IMXER  IWORK(16)  Index of the component of largest magnitude in
C                       the weighted local error vector ( e(i)/EWT(i) ),
C                       on an error return with ISTATE = -4 or -5.
C     LENRW  IWORK(17)  Length of RWORK actually required.  This is
C                       defined on normal returns and on an illegal
C                       input return for insufficient storage.
C     LENIW  IWORK(18)  Length of IWORK actually required.  This is
C                       defined on normal returns and on an illegal
C                       input return for insufficient storage.
C
C     The following two arrays are segments of the RWORK array which may
C     also be of interest to the user as optional outputs.  For each
C     array, the table below gives its internal name, its base address
C     in RWORK, and its description.
C
C     Name  Base address  Description
C     ----  ------------  ------------------------------------------------
C     YH    21            The Nordsieck history array, of size NYH by
C                         (NQCUR + 1), where NYH is the initial value of
C                         NEQ.  For j = 0,1,...,NQCUR, column j + 1 of
C                         YH contains HCUR**j/factorial(j) times the jth
C                         derivative of the interpolating polynomial
C                         currently representing the solution, evaluated
C                         at t = TCUR.
C     ACOR  LENRW-NEQ+1   Array of size NEQ used for the accumulated
```

```
C                        corrections on each step, scaled on output to
C                        represent the estimated local error in Y on
C                        the last step.  This is the vector e in the
C                        description of the error control.  It is
C                        defined only on successful return from DLSODE.
C
C
C                   Part 2.  Other Callable Routines
C                   --------------------------------
C
C     The following are optional calls which the user may make to gain
C     additional capabilities in conjunction with DLSODE.
C
C     Form of call              Function
C     ------------------------  ----------------------------------------
C     CALL XSETUN(LUN)          Set the logical unit number, LUN, for
C                               output of messages from DLSODE, if the
C                               default is not desired.  The default
C                               value of LUN is 6. This call may be made
C                               at any time and will take effect
C                               immediately.
C     CALL XSETF(MFLAG)         Set a flag to control the printing of
C                               messages by DLSODE.  MFLAG = 0 means do
C                               not print.  (Danger:  this risks losing
C                               valuable information.)  MFLAG = 1 means
C                               print (the default).  This call may be
C                               made at any time and will take effect
C                               immediately.
C     CALL DSRCOM(RSAV,ISAV,JOB)  Saves and restores the contents of the
C                               internal COMMON blocks used by DLSODE
C                               (see Part 3 below).  RSAV must be a
C                               real array of length 218 or more, and
C                               ISAV must be an integer array of length
C                               37 or more.  JOB = 1 means save COMMON
C                               into RSAV/ISAV.  JOB = 2 means restore
C                               COMMON from same.  DSRCOM is useful if
C                               one is interrupting a run and restarting
C                               later, or alternating between two or
C                               more problems solved with DLSODE.
C     CALL DINTDY(,,,,,)        Provide derivatives of y, of various
C     (see below)               orders, at a specified point t, if
C                               desired.  It may be called only after a
C                               successful return from DLSODE.  Detailed
C                               instructions follow.
C
C     Detailed instructions for using DINTDY
C     --------------------------------------
C     The form of the CALL is:
C
C           CALL DINTDY (T, K, RWORK(21), NYH, DKY, IFLAG)
C
C     The input parameters are:
C
C     T          Value of independent variable where answers are
C                desired (normally the same as the T last returned by
C                DLSODE).  For valid results, T must lie between
C                TCUR - HU and TCUR.  (See "Optional Outputs" above
C                for TCUR and HU.)
C     K          Integer order of the derivative desired.  K must
C                satisfy 0 <= K <= NQCUR, where NQCUR is the current
C                order (see "Optional Outputs").  The capability
C                corresponding to K = 0, i.e., computing y(t), is
C                already provided by DLSODE directly.  Since
C                NQCUR >= 1, the first derivative dy/dt is always
C                available with DINTDY.
C     RWORK(21)  The base address of the history array YH.
C     NYH        Column length of YH, equal to the initial value of NEQ.
```

```
C
C     The output parameters are:
C
C     DKY        Real array of length NEQ containing the computed value
C                of the Kth derivative of y(t).
C     IFLAG      Integer flag, returned as 0 if K and T were legal,
C                -1 if K was illegal, and -2 if T was illegal.
C                On an error return, a message is also written.
C
C
C                         Part 3.  Common Blocks
C                         ----------------------
C
C     If DLSODE is to be used in an overlay situation, the user must
C     declare, in the primary overlay, the variables in:
C     (1) the call sequence to DLSODE,
C     (2) the internal COMMON block /DLS001/, of length 255
C         (218 double precision words followed by 37 integer words).
C
C     If DLSODE is used on a system in which the contents of internal
C     COMMON blocks are not preserved between calls, the user should
C     declare the above COMMON block in his main program to insure that
C     its contents are preserved.
C
C     If the solution of a given problem by DLSODE is to be interrupted
C     and then later continued, as when restarting an interrupted run or
C     alternating between two or more problems, the user should save,
C     following the return from the last DLSODE call prior to the
C     interruption, the contents of the call sequence variables and the
C     internal COMMON block, and later restore these values before the
C     next DLSODE call for that problem.   In addition, if XSETUN and/or
C     XSETF was called for non-default handling of error messages, then
C     these calls must be repeated.  To save and restore the COMMON
C     block, use subroutine DSRCOM (see Part 2 above).
C
C
C               Part 4.  Optionally Replaceable Solver Routines
C               -----------------------------------------------
C
C     Below are descriptions of two routines in the DLSODE package which
C     relate to the measurement of errors.  Either routine can be
C     replaced by a user-supplied version, if desired.  However, since
C     such a replacement may have a major impact on performance, it
C     should be done only when absolutely necessary, and only with great
C     caution.  (Note:  The means by which the package version of a
C     routine is superseded by the user's version may be system-
C     dependent.)
C
C     DEWSET
C     ------
C     The following subroutine is called just before each internal
C     integration step, and sets the array of error weights, EWT, as
C     described under ITOL/RTOL/ATOL above:
C
C           SUBROUTINE DEWSET (NEQ, ITOL, RTOL, ATOL, YCUR, EWT)
C
C     where NEQ, ITOL, RTOL, and ATOL are as in the DLSODE call
C     sequence, YCUR contains the current dependent variable vector,
C     and EWT is the array of weights set by DEWSET.
C
C     If the user supplies this subroutine, it must return in EWT(i)
C     (i = 1,...,NEQ) a positive quantity suitable for comparing errors
C     in Y(i) to.  The EWT array returned by DEWSET is passed to the
C     DVNORM routine (see below), and also used by DLSODE in the
C     computation of the optional output IMXER, the diagonal Jacobian
C     approximation, and the increments for difference quotient
C     Jacobians.
```

```
C
C     In the user-supplied version of DEWSET, it may be desirable to use
C     the current values of derivatives of y. Derivatives up to order NQ
C     are available from the history array YH, described above under
C     optional outputs.  In DEWSET, YH is identical to the YCUR array,
C     extended to NQ + 1 columns with a column length of NYH and scale
C     factors of H**j/factorial(j).  On the first call for the problem,
C     given by NST = 0, NQ is 1 and H is temporarily set to 1.0.  The
C     quantities NQ, NYH, H, and NST can be obtained by including in
C     DEWSET the statements:
C
C           DOUBLE PRECISION  RLS
C           COMMON /DLS001/ RLS(218),ILS(37)
C           NQ = ILS(33)
C           NYH = ILS(12)
C           NST = ILS(34)
C           H = RLS(212)
C
C     Thus, for example, the current value of dy/dt can be obtained as
C     YCUR(NYH+i)/H (i=1,...,NEQ) (and the division by H is unnecessary
C     when NST = 0).
C
C     DVNORM
C     ------
C     DVNORM is a real function routine which computes the weighted
C     root-mean-square norm of a vector v:
C
C        d = DVNORM (n, v, w)
C
C     where:
C     n = the length of the vector,
C     v = real array of length n containing the vector,
C     w = real array of length n containing weights,
C     d = SQRT( (1/n) * sum(v(i)*w(i))**2 ).
C
C     DVNORM is called with n = NEQ and with w(i) = 1.0/EWT(i), where
C     EWT is as set by subroutine DEWSET.
C
C     If the user supplies this function, it should return a nonnegative
C     value of DVNORM suitable for use in the error control in DLSODE.
C     None of the arguments should be altered by DVNORM.  For example, a
C     user-supplied DVNORM routine might:
C     - Substitute a max-norm of (v(i)*w(i)) for the rms-norm, or
C     - Ignore some components of v in the norm, with the effect of
C       suppressing the error control on those components of Y.
C  ----------------------------------------------------------------------
C***REFERENCES  Alan C. Hindmarsh, "ODEPACK, a systematized collection
C                 of ODE solvers", in Scientific Computing, R. S.
C                 Stepleman, et al. (Eds.), (North-Holland, Amsterdam,
C                 1983), pp. 55-64.
C***ROUTINES CALLED  DEWSET, DINTDY, DUMACH, DSTODE, DVNORM, XERRWD
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   870330  Major update by ACH.
C   890426  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890501  Many improvements to prologue.  (FNF)
C   890503  A few final corrections to prologue.  (FNF)
C   890504  Minor cosmetic changes.  (FNF)
C   890510  Corrected description of Y in Arguments section.  (FNF)
C   890517  Minor corrections to prologue.  (FNF)
C   920514  Updated with prologue edited 891025 by G. Shaw for manual.
C   920515  Converted source lines to upper case.  (FNF)
C   920603  Revised XERRWV calls using mixed upper-lower case.  (ACH)
C   920616  Revised prologue comment regarding CFT.  (ACH)
C   921116  Revised prologue comments regarding Common.  (ACH).
C   930326  Added comment about non-reentrancy.  (FNF)
```

```
C   930723  Changed R1MACH to RUMACH. (FNF)
C   930801  Removed Common variables ILLIN and NTREP (affects driver
C           logic and Common references); minor changes to prologue and
C           internal comments; changed Hollerith strings to quoted
C           strings; changed internal comments to mixed case; changed
C           dummy dimensions from 1 to *. (ACH)
C   930809  Changed to generic intrinsic names; changed names of
C           subprograms and Common blocks to SLSODE etc. (ACH)
C   930929  Eliminated use of REAL intrinsic; other minor changes. (ACH)
C   931005  Generated double precision version. (ACH)
C***END PROLOGUE  DLSODE
C
C*Internal Notes:
C
C Other Routines in the DLSODE Package.
C
C In addition to Subroutine DLSODE, the DLSODE package includes the
C following subroutines and function routines:
C  DINTDY   computes an interpolated value of the y vector at t = TOUT.
C  DSTODE   is the core integrator, which does one step of the
C           integration and the associated error control.
C  DCFODE   sets all method coefficients and test constants.
C  DPREPJ   computes and preprocesses the Jacobian matrix J = df/dy
C           and the Newton iteration matrix P = I - h*l0*J.
C  DSOLSY   manages solution of linear system in chord iteration.
C  DEWSET   sets the error weight vector EWT before each step.
C  DVNORM   computes the weighted R.M.S. norm of a vector.
C  DSRCOM   is a user-callable routine to save and restore
C           the contents of the internal Common block.
C  DGEFA and DGESL   are routines from LINPACK for solving full
C           systems of linear algebraic equations.
C  DGBFA and DGBSL   are routines from LINPACK for solving banded
C           linear systems.
C  DUMACH   computes the unit roundoff in a machine-independent manner.
C  XERRWD, XSETUN, and XSETF   handle the printing of all error
C           messages and warnings.  XERRWD is machine-dependent.
C Note.. DVNORM and DUMACH are function routines.  All the others
C are subroutines.
C
C The intrinsic routines used by DLSODE are..
C ABS, MAX, MIN, MOD, SIGN, and SQRT.
C
C**End
C
C  Declare arguments.
C
      EXTERNAL F, JAC
      INTEGER NEQ, ITOL, ITASK, ISTATE, IOPT, LRW, IWORK, LIW, MF
      DOUBLE PRECISION Y, T, TOUT, RTOL, ATOL, RWORK
      DIMENSION NEQ(*), Y(*), RTOL(*), ATOL(*), RWORK(LRW), IWORK(LIW)
C
C  Declare externals.
C
      EXTERNAL DPREPJ, DSOLSY
      DOUBLE PRECISION DUMACH, DVNORM
C
C  Declare all other variables.
C
      INTEGER INIT, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,
     1   MXSTEP, MXHNIL, NHNIL, NSLAST, NYH, IOWNS
      INTEGER ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     1   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, I1, I2, IFLAG, IMXER, KGO, LF0,
     1   LENIW, LENRW, LENWM, ML, MORD, MU, MXHNL0, MXSTP0
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION ATOLI, AYI, BIG, EWTI, H0, HMAX, HMX, RH, RTOLI,
```

```
     1   TCRIT, TDIST, TNEXT, TOL, TOLSF, TP, SIZE, SUM, W0
      DIMENSION MORD(2)
      LOGICAL IHIT
      CHARACTER*80 MSG
C-----------------------------------------------------------------------
C The following internal common block contains
C (a) variables which are local to any subroutine but whose values must
C     be preserved between calls to the routine (own variables), and
C (b) variables which are communicated between subroutines.
C The structure of the block is as follows..  All real variables are
C listed first, followed by all integers.  Within each type, the
C variables are grouped with those local to Subroutine DLSODE first,
C then those local to Subroutine DSTODE, and finally those used
C for communication.  The block is declared in subroutines
C DLSODE, DINTDY, DSTODE, DPREPJ, and DSOLSY.  Groups of variables are
C replaced by dummy arrays in the common declarations in routines
C where those variables are not used.
C-----------------------------------------------------------------------
      COMMON /DLS001/ ROWNS(209),
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     2   INIT, LYH, LEWT, LACOR, LSAVF, LWM, LIWM,
     3   MXSTEP, MXHNIL, NHNIL, NSLAST, NYH, IOWNS(6),
     4   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
      DATA  MORD(1),MORD(2)/12,5/, MXSTP0/500/, MXHNL0/10/
C-----------------------------------------------------------------------
C Block A.
C This code block is executed on every call.
C It tests ISTATE and ITASK for legality and branches appropriately.
C If ISTATE .GT. 1 but the flag INIT shows that initialization has
C not yet been done, an error return occurs.
C If ISTATE = 1 and TOUT = T, return immediately.
C-----------------------------------------------------------------------
C
C***FIRST EXECUTABLE STATEMENT  DLSODE
      IF (ISTATE .LT. 1 .OR. ISTATE .GT. 3) GO TO 601
      IF (ITASK .LT. 1 .OR. ITASK .GT. 5) GO TO 602
      IF (ISTATE .EQ. 1) GO TO 10
      IF (INIT .EQ. 0) GO TO 603
      IF (ISTATE .EQ. 2) GO TO 200
      GO TO 20
 10   INIT = 0
      IF (TOUT .EQ. T) RETURN
C-----------------------------------------------------------------------
C Block B.
C The next code block is executed for the initial call (ISTATE = 1),
C or for a continuation call with parameter changes (ISTATE = 3).
C It contains checking of all inputs and various initializations.
C
C First check legality of the non-optional inputs NEQ, ITOL, IOPT,
C MF, ML, and MU.
C-----------------------------------------------------------------------
 20   IF (NEQ(1) .LE. 0) GO TO 604
      IF (ISTATE .EQ. 1) GO TO 25
      IF (NEQ(1) .GT. N) GO TO 605
 25   N = NEQ(1)
      IF (ITOL .LT. 1 .OR. ITOL .GT. 4) GO TO 606
      IF (IOPT .LT. 0 .OR. IOPT .GT. 1) GO TO 607
      METH = MF/10
      MITER = MF - 10*METH
      IF (METH .LT. 1 .OR. METH .GT. 2) GO TO 608
      IF (MITER .LT. 0 .OR. MITER .GT. 5) GO TO 608
      IF (MITER .LE. 3) GO TO 30
      ML = IWORK(1)
      MU = IWORK(2)
      IF (ML .LT. 0 .OR. ML .GE. N) GO TO 609
```

```
      IF (MU .LT. 0 .OR. MU .GE. N) GO TO 610
 30   CONTINUE
C Next process and check the optional inputs. --------------------------
      IF (IOPT .EQ. 1) GO TO 40
      MAXORD = MORD(METH)
      MXSTEP = MXSTP0
      MXHNIL = MXHNL0
      IF (ISTATE .EQ. 1) H0 = 0.0D0
      HMXI = 0.0D0
      HMIN = 0.0D0
      GO TO 60
 40   MAXORD = IWORK(5)
      IF (MAXORD .LT. 0) GO TO 611
      IF (MAXORD .EQ. 0) MAXORD = 100
      MAXORD = MIN(MAXORD,MORD(METH))
      MXSTEP = IWORK(6)
      IF (MXSTEP .LT. 0) GO TO 612
      IF (MXSTEP .EQ. 0) MXSTEP = MXSTP0
      MXHNIL = IWORK(7)
      IF (MXHNIL .LT. 0) GO TO 613
      IF (MXHNIL .EQ. 0) MXHNIL = MXHNL0
      IF (ISTATE .NE. 1) GO TO 50
      H0 = RWORK(5)
      IF ((TOUT - T)*H0 .LT. 0.0D0) GO TO 614
 50   HMAX = RWORK(6)
      IF (HMAX .LT. 0.0D0) GO TO 615
      HMXI = 0.0D0
      IF (HMAX .GT. 0.0D0) HMXI = 1.0D0/HMAX
      HMIN = RWORK(7)
      IF (HMIN .LT. 0.0D0) GO TO 616
C-----------------------------------------------------------------------
C Set work array pointers and check lengths LRW and LIW.
C Pointers to segments of RWORK and IWORK are named by prefixing L to
C the name of the segment.  E.g., the segment YH starts at RWORK(LYH).
C Segments of RWORK (in order) are denoted  YH, WM, EWT, SAVF, ACOR.
C-----------------------------------------------------------------------
 60   LYH = 21
      IF (ISTATE .EQ. 1) NYH = N
      LWM = LYH + (MAXORD + 1)*NYH
      IF (MITER .EQ. 0) LENWM = 0
      IF (MITER .EQ. 1 .OR. MITER .EQ. 2) LENWM = N*N + 2
      IF (MITER .EQ. 3) LENWM = N + 2
      IF (MITER .GE. 4) LENWM = (2*ML + MU + 1)*N + 2
      LEWT = LWM + LENWM
      LSAVF = LEWT + N
      LACOR = LSAVF + N
      LENRW = LACOR + N - 1
      IWORK(17) = LENRW
      LIWM = 1
      LENIW = 20 + N
      IF (MITER .EQ. 0 .OR. MITER .EQ. 3) LENIW = 20
      IWORK(18) = LENIW
      IF (LENRW .GT. LRW) GO TO 617
      IF (LENIW .GT. LIW) GO TO 618
C Check RTOL and ATOL for legality. ------------------------------------
      RTOLI = RTOL(1)
      ATOLI = ATOL(1)
      DO 70 I = 1,N
        IF (ITOL .GE. 3) RTOLI = RTOL(I)
        IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)
        IF (RTOLI .LT. 0.0D0) GO TO 619
        IF (ATOLI .LT. 0.0D0) GO TO 620
 70     CONTINUE
      IF (ISTATE .EQ. 1) GO TO 100
C If ISTATE = 3, set flag to signal parameter changes to DSTODE. -------
      JSTART = -1
      IF (NQ .LE. MAXORD) GO TO 90
```

```
C MAXORD was reduced below NQ.  Copy YH(*,MAXORD+2) into SAVF. ---------
      DO 80 I = 1,N
 80     RWORK(I+LSAVF-1) = RWORK(I+LWM-1)
C Reload WM(1) = RWORK(LWM), since LWM may have changed. ---------------
 90   IF (MITER .GT. 0) RWORK(LWM) = SQRT(UROUND)
      IF (N .EQ. NYH) GO TO 200
C NEQ was reduced.  Zero part of YH to avoid undefined references. -----
      I1 = LYH + L*NYH
      I2 = LYH + (MAXORD + 1)*NYH - 1
      IF (I1 .GT. I2) GO TO 200
      DO 95 I = I1,I2
 95     RWORK(I) = 0.0D0
      GO TO 200
C-----------------------------------------------------------------------
C Block C.
C The next block is for the initial call only (ISTATE = 1).
C It contains all remaining initializations, the initial call to F,
C and the calculation of the initial step size.
C The error weights in EWT are inverted after being loaded.
C-----------------------------------------------------------------------
 100  UROUND = DUMACH()
      TN = T
      IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 110
      TCRIT = RWORK(1)
      IF ((TCRIT - TOUT)*(TOUT - T) .LT. 0.0D0) GO TO 625
      IF (H0 .NE. 0.0D0 .AND. (T + H0 - TCRIT)*H0 .GT. 0.0D0)
     1   H0 = TCRIT - T
 110  JSTART = 0
      IF (MITER .GT. 0) RWORK(LWM) = SQRT(UROUND)
      NHNIL = 0
      NST = 0
      NJE = 0
      NSLAST = 0
      HU = 0.0D0
      NQU = 0
      CCMAX = 0.3D0
      MAXCOR = 3
      MSBP = 20
      MXNCF = 10
C Initial call to F.  (LF0 points to YH(*,2).) -------------------------
      LF0 = LYH + NYH
      CALL F (NEQ, T, Y, RWORK(LF0))
      NFE = 1
C Load the initial value vector in YH. ---------------------------------
      DO 115 I = 1,N
 115    RWORK(I+LYH-1) = Y(I)
C Load and invert the EWT array.  (H is temporarily set to 1.0.) -------
      NQ = 1
      H = 1.0D0
      CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))
      DO 120 I = 1,N
        IF (RWORK(I+LEWT-1) .LE. 0.0D0) GO TO 621
 120    RWORK(I+LEWT-1) = 1.0D0/RWORK(I+LEWT-1)
C-----------------------------------------------------------------------
C The coding below computes the step size, H0, to be attempted on the
C first step, unless the user has supplied a value for this.
C First check that TOUT - T differs significantly from zero.
C A scalar tolerance quantity TOL is computed, as MAX(RTOL(I))
C if this is positive, or MAX(ATOL(I)/ABS(Y(I))) otherwise, adjusted
C so as to be between 100*UROUND and 1.0E-3.
C Then the computed value H0 is given by..
C                                     NEQ
C   H0**2 = TOL / ( w0**-2 + (1/NEQ) * SUM ( f(i)/ywt(i) )**2  )
C                                      1
C where   w0     = MAX ( ABS(T), ABS(TOUT) ),
C         f(i)   = i-th component of initial value of f,
C         ywt(i) = EWT(i)/TOL  (a weight for y(i)).
```

```
C The sign of H0 is inferred from the initial values of TOUT and T.
C-----------------------------------------------------------------------
      IF (H0 .NE. 0.0D0) GO TO 180
      TDIST = ABS(TOUT - T)
      W0 = MAX(ABS(T),ABS(TOUT))
      IF (TDIST .LT. 2.0D0*UROUND*W0) GO TO 622
      TOL = RTOL(1)
      IF (ITOL .LE. 2) GO TO 140
      DO 130 I = 1,N
 130    TOL = MAX(TOL,RTOL(I))
 140  IF (TOL .GT. 0.0D0) GO TO 160
      ATOLI = ATOL(1)
      DO 150 I = 1,N
        IF (ITOL .EQ. 2 .OR. ITOL .EQ. 4) ATOLI = ATOL(I)
        AYI = ABS(Y(I))
        IF (AYI .NE. 0.0D0) TOL = MAX(TOL,ATOLI/AYI)
 150    CONTINUE
 160  TOL = MAX(TOL,100.0D0*UROUND)
      TOL = MIN(TOL,0.001D0)
      SUM = DVNORM (N, RWORK(LF0), RWORK(LEWT))
      SUM = 1.0D0/(TOL*W0*W0) + TOL*SUM**2
      H0 = 1.0D0/SQRT(SUM)
      H0 = MIN(H0,TDIST)
      H0 = SIGN(H0,TOUT-T)
C Adjust H0 if necessary to meet HMAX bound. ---------------------------
 180  RH = ABS(H0)*HMXI
      IF (RH .GT. 1.0D0) H0 = H0/RH
C Load H with H0 and scale YH(*,2) by H0. ------------------------------
      H = H0
      DO 190 I = 1,N
 190    RWORK(I+LF0-1) = H0*RWORK(I+LF0-1)
      GO TO 270
C-----------------------------------------------------------------------
C Block D.
C The next code block is for continuation calls only (ISTATE = 2 or 3)
C and is to check stop conditions before taking a step.
C-----------------------------------------------------------------------
 200  NSLAST = NST
      GO TO (210, 250, 220, 230, 240), ITASK
 210  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      IF (IFLAG .NE. 0) GO TO 627
      T = TOUT
      GO TO 420
 220  TP = TN - HU*(1.0D0 + 100.0D0*UROUND)
      IF ((TP - TOUT)*H .GT. 0.0D0) GO TO 623
      IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      GO TO 400
 230  TCRIT = RWORK(1)
      IF ((TN - TCRIT)*H .GT. 0.0D0) GO TO 624
      IF ((TCRIT - TOUT)*H .LT. 0.0D0) GO TO 625
      IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 245
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      IF (IFLAG .NE. 0) GO TO 627
      T = TOUT
      GO TO 420
 240  TCRIT = RWORK(1)
      IF ((TN - TCRIT)*H .GT. 0.0D0) GO TO 624
 245  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. 100.0D0*UROUND*HMX
      IF (IHIT) GO TO 400
      TNEXT = TN + H*(1.0D0 + 4.0D0*UROUND)
      IF ((TNEXT - TCRIT)*H .LE. 0.0D0) GO TO 250
      H = (TCRIT - TN)*(1.0D0 - 4.0D0*UROUND)
      IF (ISTATE .EQ. 2) JSTART = -2
C-----------------------------------------------------------------------
C Block E.
```

```
C The next block is normally executed for all calls and contains
C the call to the one-step core integrator DSTODE.
C
C This is a looping point for the integration steps.
C
C First check for too many steps being taken, update EWT (if not at
C start of problem), check for too much accuracy being requested, and
C check for H below the roundoff level in T.
C-----------------------------------------------------------------------
 250  CONTINUE
      IF ((NST-NSLAST) .GE. MXSTEP) GO TO 500
      CALL DEWSET (N, ITOL, RTOL, ATOL, RWORK(LYH), RWORK(LEWT))
      DO 260 I = 1,N
        IF (RWORK(I+LEWT-1) .LE. 0.0D0) GO TO 510
 260    RWORK(I+LEWT-1) = 1.0D0/RWORK(I+LEWT-1)
 270  TOLSF = UROUND*DVNORM (N, RWORK(LYH), RWORK(LEWT))
      IF (TOLSF .LE. 1.0D0) GO TO 280
      TOLSF = TOLSF*2.0D0
      IF (NST .EQ. 0) GO TO 626
      GO TO 520
 280  IF ((TN + H) .NE. TN) GO TO 290
      NHNIL = NHNIL + 1
      IF (NHNIL .GT. MXHNIL) GO TO 290
      MSG = 'DLSODE-  Warning..internal T (=R1) and H (=R2) are'
      CALL XERRWD (MSG, 50, 101, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='      such that in the machine, T + H = T on the next step  '
      CALL XERRWD (MSG, 60, 101, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      (H = step size). Solver will continue anyway'
      CALL XERRWD (MSG, 50, 101, 0, 0, 0, 0, 2, TN, H)
      IF (NHNIL .LT. MXHNIL) GO TO 290
      MSG = 'DLSODE-  Above warning has been issued I1 times.  '
      CALL XERRWD (MSG, 50, 102, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      It will not be issued again for this problem'
      CALL XERRWD (MSG, 50, 102, 0, 1, MXHNIL, 0, 0, 0.0D0, 0.0D0)
 290  CONTINUE
C-----------------------------------------------------------------------
C  CALL DSTODE(NEQ,Y,YH,NYH,YH,EWT,SAVF,ACOR,WM,IWM,F,JAC,DPREPJ,DSOLSY)
C-----------------------------------------------------------------------
      CALL DSTODE (NEQ, Y, RWORK(LYH), NYH, RWORK(LYH), RWORK(LEWT),
     1   RWORK(LSAVF), RWORK(LACOR), RWORK(LWM), IWORK(LIWM),
     2   F, JAC, DPREPJ, DSOLSY)
      KGO = 1 - KFLAG
      GO TO (300, 530, 540), KGO
C-----------------------------------------------------------------------
C Block F.
C The following block handles the case of a successful return from the
C core integrator (KFLAG = 0).  Test for stop conditions.
C-----------------------------------------------------------------------
 300  INIT = 1
      GO TO (310, 400, 330, 340, 350), ITASK
C ITASK = 1.  If TOUT has been reached, interpolate. -------------------
 310  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 250
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      T = TOUT
      GO TO 420
C ITASK = 3.  Jump to exit if TOUT was reached. ------------------------
 330  IF ((TN - TOUT)*H .GE. 0.0D0) GO TO 400
      GO TO 250
C ITASK = 4.  See if TOUT or TCRIT was reached.  Adjust H if necessary.
 340  IF ((TN - TOUT)*H .LT. 0.0D0) GO TO 345
      CALL DINTDY (TOUT, 0, RWORK(LYH), NYH, Y, IFLAG)
      T = TOUT
      GO TO 420
 345  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. 100.0D0*UROUND*HMX
      IF (IHIT) GO TO 400
      TNEXT = TN + H*(1.0D0 + 4.0D0*UROUND)
```

```
      IF ((TNEXT - TCRIT)*H .LE. 0.0D0) GO TO 250
      H = (TCRIT - TN)*(1.0D0 - 4.0D0*UROUND)
      JSTART = -2
      GO TO 250
C ITASK = 5.  See if TCRIT was reached and jump to exit. ---------------
 350  HMX = ABS(TN) + ABS(H)
      IHIT = ABS(TN - TCRIT) .LE. 100.0D0*UROUND*HMX
C-----------------------------------------------------------------------
C Block G.
C The following block handles all successful returns from DLSODE.
C If ITASK .NE. 1, Y is loaded from YH and T is set accordingly.
C ISTATE is set to 2, the illegal input counter is zeroed, and the
C optional outputs are loaded into the work arrays before returning.
C If ISTATE = 1 and TOUT = T, there is a return with no action taken.
C-----------------------------------------------------------------------
 400  DO 410 I = 1,N
 410    Y(I) = RWORK(I+LYH-1)
      T = TN
      IF (ITASK .NE. 4 .AND. ITASK .NE. 5) GO TO 420
      IF (IHIT) T = TCRIT
 420  ISTATE = 2
      RWORK(11) = HU
      RWORK(12) = H
      RWORK(13) = TN
      IWORK(11) = NST
      IWORK(12) = NFE
      IWORK(13) = NJE
      IWORK(14) = NQU
      IWORK(15) = NQ
      RETURN
C-----------------------------------------------------------------------
C Block H.
C The following block handles all unsuccessful returns other than
C those for illegal input.  First the error message routine is called.
C If there was an error test or convergence test failure, IMXER is set.
C Then Y is loaded from YH and T is set to TN.  The optional outputs
C are loaded into the work arrays before returning.
C-----------------------------------------------------------------------
C The maximum number of steps was taken before reaching TOUT. ----------
 500  MSG = 'DLSODE-  At current T (=R1), MXSTEP (=I1) steps   '
      CALL XERRWD (MSG, 50, 201, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      taken on this call before reaching TOUT     '
      CALL XERRWD (MSG, 50, 201, 0, 1, MXSTEP, 0, 1, TN, 0.0D0)
      ISTATE = -1
      GO TO 580
C EWT(I) .LE. 0.0 for some I (not at start of problem). ----------------
 510  EWTI = RWORK(LEWT+I-1)
      MSG = 'DLSODE-  At T (=R1), EWT(I1) has become R2 .LE. 0.'
      CALL XERRWD (MSG, 50, 202, 0, 1, I, 0, 2, TN, EWTI)
      ISTATE = -6
      GO TO 580
C Too much accuracy requested for machine precision. -------------------
 520  MSG = 'DLSODE-  At T (=R1), too much accuracy requested  '
      CALL XERRWD (MSG, 50, 203, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      for precision of machine..  see TOLSF (=R2) '
      CALL XERRWD (MSG, 50, 203, 0, 0, 0, 0, 2, TN, TOLSF)
      RWORK(14) = TOLSF
      ISTATE = -2
      GO TO 580
C KFLAG = -1.  Error test failed repeatedly or with ABS(H) = HMIN. -----
 530  MSG = 'DLSODE-  At T(=R1) and step size H(=R2), the error'
      CALL XERRWD (MSG, 50, 204, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      test failed repeatedly or with ABS(H) = HMIN'
      CALL XERRWD (MSG, 50, 204, 0, 0, 0, 0, 2, TN, H)
      ISTATE = -4
      GO TO 560
C KFLAG = -2.  Convergence failed repeatedly or with ABS(H) = HMIN. ----
```

```
 540  MSG = 'DLSODE-  At T (=R1) and step size H (=R2), the    '
      CALL XERRWD (MSG, 50, 205, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      corrector convergence failed repeatedly     '
      CALL XERRWD (MSG, 50, 205, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG = '      or with ABS(H) = HMIN   '
      CALL XERRWD (MSG, 30, 205, 0, 0, 0, 0, 2, TN, H)
      ISTATE = -5
C Compute IMXER if relevant. -------------------------------------------
 560  BIG = 0.0D0
      IMXER = 1
      DO 570 I = 1,N
        SIZE = ABS(RWORK(I+LACOR-1)*RWORK(I+LEWT-1))
        IF (BIG .GE. SIZE) GO TO 570
        BIG = SIZE
        IMXER = I
 570    CONTINUE
      IWORK(16) = IMXER
C Set Y vector, T, and optional outputs. -------------------------------
 580  DO 590 I = 1,N
 590    Y(I) = RWORK(I+LYH-1)
      T = TN
      RWORK(11) = HU
      RWORK(12) = H
      RWORK(13) = TN
      IWORK(11) = NST
      IWORK(12) = NFE
      IWORK(13) = NJE
      IWORK(14) = NQU
      IWORK(15) = NQ
      RETURN
C-----------------------------------------------------------------------
C Block I.
C The following block handles all error returns due to illegal input
C (ISTATE = -3), as detected before calling the core integrator.
C First the error message routine is called.  If the illegal input
C is a negative ISTATE, the run is aborted (apparent infinite loop).
C-----------------------------------------------------------------------
 601  MSG = 'DLSODE-  ISTATE (=I1) illegal '
      CALL XERRWD (MSG, 30, 1, 0, 1, ISTATE, 0, 0, 0.0D0, 0.0D0)
      IF (ISTATE .LT. 0) GO TO 800
      GO TO 700
 602  MSG = 'DLSODE-  ITASK (=I1) illegal  '
      CALL XERRWD (MSG, 30, 2, 0, 1, ITASK, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 603  MSG = 'DLSODE-  ISTATE .GT. 1 but DLSODE not initialized '
      CALL XERRWD (MSG, 50, 3, 0, 0, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 604  MSG = 'DLSODE-  NEQ (=I1) .LT. 1     '
      CALL XERRWD (MSG, 30, 4, 0, 1, NEQ(1), 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 605  MSG = 'DLSODE-  ISTATE = 3 and NEQ increased (I1 to I2)  '
      CALL XERRWD (MSG, 50, 5, 0, 2, N, NEQ(1), 0, 0.0D0, 0.0D0)
      GO TO 700
 606  MSG = 'DLSODE-  ITOL (=I1) illegal   '
      CALL XERRWD (MSG, 30, 6, 0, 1, ITOL, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 607  MSG = 'DLSODE-  IOPT (=I1) illegal   '
      CALL XERRWD (MSG, 30, 7, 0, 1, IOPT, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 608  MSG = 'DLSODE-  MF (=I1) illegal     '
      CALL XERRWD (MSG, 30, 8, 0, 1, MF, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 609  MSG = 'DLSODE-  ML (=I1) illegal.. .LT.0 or .GE.NEQ (=I2)'
      CALL XERRWD (MSG, 50, 9, 0, 2, ML, NEQ(1), 0, 0.0D0, 0.0D0)
      GO TO 700
 610  MSG = 'DLSODE-  MU (=I1) illegal.. .LT.0 or .GE.NEQ (=I2)'
      CALL XERRWD (MSG, 50, 10, 0, 2, MU, NEQ(1), 0, 0.0D0, 0.0D0)
```

```
      GO TO 700
 611  MSG = 'DLSODE-  MAXORD (=I1) .LT. 0  '
      CALL XERRWD (MSG, 30, 11, 0, 1, MAXORD, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 612  MSG = 'DLSODE-  MXSTEP (=I1) .LT. 0  '
      CALL XERRWD (MSG, 30, 12, 0, 1, MXSTEP, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 613  MSG = 'DLSODE-  MXHNIL (=I1) .LT. 0  '
      CALL XERRWD (MSG, 30, 13, 0, 1, MXHNIL, 0, 0, 0.0D0, 0.0D0)
      GO TO 700
 614  MSG = 'DLSODE-  TOUT (=R1) behind T (=R2)      '
      CALL XERRWD (MSG, 40, 14, 0, 0, 0, 0, 2, TOUT, T)
      MSG = '      Integration direction is given by H0 (=R1)  '
      CALL XERRWD (MSG, 50, 14, 0, 0, 0, 0, 1, H0, 0.0D0)
      GO TO 700
 615  MSG = 'DLSODE-  HMAX (=R1) .LT. 0.0  '
      CALL XERRWD (MSG, 30, 15, 0, 0, 0, 0, 1, HMAX, 0.0D0)
      GO TO 700
 616  MSG = 'DLSODE-  HMIN (=R1) .LT. 0.0  '
      CALL XERRWD (MSG, 30, 16, 0, 0, 0, 0, 1, HMIN, 0.0D0)
      GO TO 700
 617  CONTINUE
      MSG='DLSODE-  RWORK length needed, LENRW (=I1), exceeds LRW (=I2)'
      CALL XERRWD (MSG, 60, 17, 0, 2, LENRW, LRW, 0, 0.0D0, 0.0D0)
      GO TO 700
 618  CONTINUE
      MSG='DLSODE-  IWORK length needed, LENIW (=I1), exceeds LIW (=I2)'
      CALL XERRWD (MSG, 60, 18, 0, 2, LENIW, LIW, 0, 0.0D0, 0.0D0)
      GO TO 700
 619  MSG = 'DLSODE-  RTOL(I1) is R1 .LT. 0.0        '
      CALL XERRWD (MSG, 40, 19, 0, 1, I, 0, 1, RTOLI, 0.0D0)
      GO TO 700
 620  MSG = 'DLSODE-  ATOL(I1) is R1 .LT. 0.0        '
      CALL XERRWD (MSG, 40, 20, 0, 1, I, 0, 1, ATOLI, 0.0D0)
      GO TO 700
 621  EWTI = RWORK(LEWT+I-1)
      MSG = 'DLSODE-  EWT(I1) is R1 .LE. 0.0         '
      CALL XERRWD (MSG, 40, 21, 0, 1, I, 0, 1, EWTI, 0.0D0)
      GO TO 700
 622  CONTINUE
      MSG='DLSODE-  TOUT (=R1) too close to T(=R2) to start integration'
      CALL XERRWD (MSG, 60, 22, 0, 0, 0, 0, 2, TOUT, T)
      GO TO 700
 623  CONTINUE
      MSG='DLSODE-  ITASK = I1 and TOUT (=R1) behind TCUR - HU (= R2)  '
      CALL XERRWD (MSG, 60, 23, 0, 1, ITASK, 0, 2, TOUT, TP)
      GO TO 700
 624  CONTINUE
      MSG='DLSODE-  ITASK = 4 OR 5 and TCRIT (=R1) behind TCUR (=R2)   '
      CALL XERRWD (MSG, 60, 24, 0, 0, 0, 0, 2, TCRIT, TN)
      GO TO 700
 625  CONTINUE
      MSG='DLSODE-  ITASK = 4 or 5 and TCRIT (=R1) behind TOUT (=R2)   '
      CALL XERRWD (MSG, 60, 25, 0, 0, 0, 0, 2, TCRIT, TOUT)
      GO TO 700
 626  MSG = 'DLSODE-  At start of problem, too much accuracy   '
      CALL XERRWD (MSG, 50, 26, 0, 0, 0, 0, 0, 0.0D0, 0.0D0)
      MSG='      requested for precision of machine..  See TOLSF (=R1) '
      CALL XERRWD (MSG, 60, 26, 0, 0, 0, 0, 1, TOLSF, 0.0D0)
      RWORK(14) = TOLSF
      GO TO 700
 627  MSG = 'DLSODE-  Trouble in DINTDY.  ITASK = I1, TOUT = R1'
      CALL XERRWD (MSG, 50, 27, 0, 1, ITASK, 0, 1, TOUT, 0.0D0)
C
 700  ISTATE = -3
      RETURN
C
```

```
 800  MSG = 'DLSODE-  Run aborted.. apparent infinite loop     '
      CALL XERRWD (MSG, 50, 303, 2, 0, 0, 0, 0, 0.0D0, 0.0D0)
      RETURN
C----------------------- END OF SUBROUTINE DLSODE ----------------------
      END
*DECK DCFODE
      SUBROUTINE DCFODE (METH, ELCO, TESCO)
C***BEGIN PROLOGUE  DCFODE
C***SUBSIDIARY
C***PURPOSE  Set ODE integrator coefficients.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SCFODE-S, DCFODE-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DCFODE is called by the integrator routine to set coefficients
C  needed there.  The coefficients for the current method, as
C  given by the value of METH, are set for all orders and saved.
C  The maximum order assumed here is 12 if METH = 1 and 5 if METH = 2.
C  (A smaller value of the maximum order is also allowed.)
C  DCFODE is called once at the beginning of the problem,
C  and is not called again unless and until METH is changed.
C
C  The ELCO array contains the basic method coefficients.
C  The coefficients el(i), 1 .le. i .le. nq+1, for the method of
C  order nq are stored in ELCO(i,nq).  They are given by a genetrating
C  polynomial, i.e.,
C      l(x) = el(1) + el(2)*x + ... + el(nq+1)*x**nq.
C  For the implicit Adams methods, l(x) is given by
C      dl/dx = (x+1)*(x+2)*...*(x+nq-1)/factorial(nq-1),    l(-1) = 0.
C  For the BDF methods, l(x) is given by
C      l(x) = (x+1)*(x+2)* ... *(x+nq)/K,
C  where         K = factorial(nq)*(1 + 1/2 + ... + 1/nq).
C
C  The TESCO array contains test constants used for the
C  local error test and the selection of step size and/or order.
C  At order nq, TESCO(k,nq) is used for the selection of step
C  size at order nq - 1 if k = 1, at order nq if k = 2, and at order
C  nq + 1 if k = 3.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DCFODE
C**End
      INTEGER METH
      INTEGER I, IB, NQ, NQM1, NQP1
      DOUBLE PRECISION ELCO, TESCO
      DOUBLE PRECISION AGAMQ, FNQ, FNQM1, PC, PINT, RAGQ,
     1   RQFAC, RQ1FAC, TSIGN, XPIN
      DIMENSION ELCO(13,12), TESCO(3,12)
      DIMENSION PC(12)
C
C***FIRST EXECUTABLE STATEMENT  DCFODE
      GO TO (100, 200), METH
C
 100  ELCO(1,1) = 1.0D0
      ELCO(2,1) = 1.0D0
      TESCO(1,1) = 0.0D0
      TESCO(2,1) = 2.0D0
      TESCO(1,2) = 1.0D0
      TESCO(3,12) = 0.0D0
      PC(1) = 1.0D0
```

```
      RQFAC = 1.0D0
      DO 140 NQ = 2,12
C-----------------------------------------------------------------------
C The PC array will contain the coefficients of the polynomial
C     p(x) = (x+1)*(x+2)*...*(x+nq-1).
C Initially, p(x) = 1.
C-----------------------------------------------------------------------
        RQ1FAC = RQFAC
        RQFAC = RQFAC/NQ
        NQM1 = NQ - 1
        FNQM1 = NQM1
        NQP1 = NQ + 1
C Form coefficients of p(x)*(x+nq-1). ----------------------------------
        PC(NQ) = 0.0D0
        DO 110 IB = 1,NQM1
          I = NQP1 - IB
 110      PC(I) = PC(I-1) + FNQM1*PC(I)
        PC(1) = FNQM1*PC(1)
C Compute integral, -1 to 0, of p(x) and x*p(x). -----------------------
        PINT = PC(1)
        XPIN = PC(1)/2.0D0
        TSIGN = 1.0D0
        DO 120 I = 2,NQ
          TSIGN = -TSIGN
          PINT = PINT + TSIGN*PC(I)/I
 120      XPIN = XPIN + TSIGN*PC(I)/(I+1)
C Store coefficients in ELCO and TESCO. --------------------------------
        ELCO(1,NQ) = PINT*RQ1FAC
        ELCO(2,NQ) = 1.0D0
        DO 130 I = 2,NQ
 130      ELCO(I+1,NQ) = RQ1FAC*PC(I)/I
        AGAMQ = RQFAC*XPIN
        RAGQ = 1.0D0/AGAMQ
        TESCO(2,NQ) = RAGQ
        IF (NQ .LT. 12) TESCO(1,NQP1) = RAGQ*RQFAC/NQP1
        TESCO(3,NQM1) = RAGQ
 140    CONTINUE
      RETURN
C
 200  PC(1) = 1.0D0
      RQ1FAC = 1.0D0
      DO 230 NQ = 1,5
C-----------------------------------------------------------------------
C The PC array will contain the coefficients of the polynomial
C     p(x) = (x+1)*(x+2)*...*(x+nq).
C Initially, p(x) = 1.
C-----------------------------------------------------------------------
        FNQ = NQ
        NQP1 = NQ + 1
C form coefficients of p(x)*(x+nq). ------------------------------------
        PC(NQP1) = 0.0D0
        DO 210 IB = 1,NQ
          I = NQ + 2 - IB
 210      PC(I) = PC(I-1) + FNQ*PC(I)
        PC(1) = FNQ*PC(1)
C Store coefficients in ELCO and TESCO. --------------------------------
        DO 220 I = 1,NQP1
 220      ELCO(I,NQ) = PC(I)/PC(2)
        ELCO(2,NQ) = 1.0D0
        TESCO(1,NQ) = RQ1FAC
        TESCO(2,NQ) = NQP1/ELCO(1,NQ)
        TESCO(3,NQ) = (NQ+2)/ELCO(1,NQ)
        RQ1FAC = RQ1FAC/FNQ
 230    CONTINUE
      RETURN
C----------------------- END OF SUBROUTINE DCFODE ----------------------
      END
```

```
*DECK DINTDY
      SUBROUTINE DINTDY (T, K, YH, NYH, DKY, IFLAG)
C***BEGIN PROLOGUE  DINTDY
C***SUBSIDIARY
C***PURPOSE  Interpolate solution derivatives.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SINTDY-S, DINTDY-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DINTDY computes interpolated values of the K-th derivative of the
C  dependent variable vector y, and stores it in DKY.  This routine
C  is called within the package with K = 0 and T = TOUT, but may
C  also be called by the user for any K up to the current order.
C  (See detailed instructions in the usage documentation.)
C
C  The computed values in DKY are gotten by interpolation using the
C  Nordsieck history array YH.  This array corresponds uniquely to a
C  vector-valued polynomial of degree NQCUR or less, and DKY is set
C  to the K-th derivative of this polynomial at T.
C  The formula for DKY is:
C               q
C   DKY(i)  =  sum  c(j,K) * (T - tn)**(j-K) * h**(-j) * YH(i,j+1)
C              j=K
C  where  c(j,K) = j*(j-1)*...*(j-K+1), q = NQCUR, tn = TCUR, h = HCUR.
C  The quantities  nq = NQCUR, l = nq+1, N = NEQ, tn, and h are
C  communicated by COMMON.  The above sum is done in reverse order.
C  IFLAG is returned negative if either K or T is out of bounds.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  XERRWD
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DINTDY
C**End
      INTEGER K, NYH, IFLAG
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     2   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, IC, J, JB, JB2, JJ, JJ1, JP1
      DOUBLE PRECISION T, YH, DKY
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION C, R, S, TP
      CHARACTER*80 MSG
      DIMENSION YH(NYH,*), DKY(*)
      COMMON /DLS001/ ROWNS(209),
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     3   IOWND(12), IOWNS(6),
     4   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
C***FIRST EXECUTABLE STATEMENT  DINTDY
      IFLAG = 0
      IF (K .LT. 0 .OR. K .GT. NQ) GO TO 80
      TP = TN - HU -  100.0D0*UROUND*(TN + HU)
      IF ((T-TP)*(T-TN) .GT. 0.0D0) GO TO 90
C
      S = (T - TN)/H
      IC = 1
      IF (K .EQ. 0) GO TO 15
      JJ1 = L - K
      DO 10 JJ = JJ1,NQ
```

```
 10     IC = IC*JJ
 15   C = IC
      DO 20 I = 1,N
 20     DKY(I) = C*YH(I,L)
      IF (K .EQ. NQ) GO TO 55
      JB2 = NQ - K
      DO 50 JB = 1,JB2
        J = NQ - JB
        JP1 = J + 1
        IC = 1
        IF (K .EQ. 0) GO TO 35
        JJ1 = JP1 - K
        DO 30 JJ = JJ1,J
 30       IC = IC*JJ
 35     C = IC
        DO 40 I = 1,N
 40       DKY(I) = C*YH(I,JP1) + S*DKY(I)
 50     CONTINUE
      IF (K .EQ. 0) RETURN
 55   R = H**(-K)
      DO 60 I = 1,N
 60     DKY(I) = R*DKY(I)
      RETURN
C
 80   MSG = 'DINTDY-  K (=I1) illegal      '
      CALL XERRWD (MSG, 30, 51, 0, 1, K, 0, 0, 0.0D0, 0.0D0)
      IFLAG = -1
      RETURN
 90   MSG = 'DINTDY-  T (=R1) illegal      '
      CALL XERRWD (MSG, 30, 52, 0, 0, 0, 0, 1, T, 0.0D0)
      MSG='      T not in interval TCUR - HU (= R1) to TCUR (=R2)      '
      CALL XERRWD (MSG, 60, 52, 0, 0, 0, 0, 2, TP, TN)
      IFLAG = -2
      RETURN
C----------------------- END OF SUBROUTINE DINTDY ----------------------
      END
*DECK DPREPJ
      SUBROUTINE DPREPJ (NEQ, Y, YH, NYH, EWT, FTEM, SAVF, WM, IWM,
     1   F, JAC)
C***BEGIN PROLOGUE  DPREPJ
C***SUBSIDIARY
C***PURPOSE  Compute and process Newton iteration matrix.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SPREPJ-S, DPREPJ-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DPREPJ is called by DSTODE to compute and process the matrix
C  P = I - h*el(1)*J , where J is an approximation to the Jacobian.
C  Here J is computed by the user-supplied routine JAC if
C  MITER = 1 or 4, or by finite differencing if MITER = 2, 3, or 5.
C  If MITER = 3, a diagonal approximation to J is used.
C  J is stored in WM and replaced by P.  If MITER .ne. 3, P is then
C  subjected to LU decomposition in preparation for later solution
C  of linear systems with P as coefficient matrix.  This is done
C  by DGEFA if MITER = 1 or 2, and by DGBFA if MITER = 4 or 5.
C
C  In addition to variables described in DSTODE and DLSODE prologues,
C  communication with DPREPJ uses the following:
C  Y     = array containing predicted values on entry.
C  FTEM  = work array of length N (ACOR in DSTODE).
C  SAVF  = array containing f evaluated at predicted y.
C  WM    = real work space for matrices.  On output it contains the
C          inverse diagonal matrix if MITER = 3 and the LU decomposition
C          of P if MITER is 1, 2 , 4, or 5.
C          Storage of matrix elements starts at WM(3).
C          WM also contains the following matrix-related data:
```

```
C          WM(1) = SQRT(UROUND), used in numerical Jacobian increments.
C          WM(2) = H*EL0, saved for later use if MITER = 3.
C  IWM   = integer work space containing pivot information, starting at
C          IWM(21), if MITER is 1, 2, 4, or 5.  IWM also contains band
C          parameters ML = IWM(1) and MU = IWM(2) if MITER is 4 or 5.
C  EL0   = EL(1) (input).
C  IERPJ = output error flag,  = 0 if no trouble, .gt. 0 if
C          P matrix found to be singular.
C  JCUR  = output flag = 1 to indicate that the Jacobian matrix
C          (or approximation) is now current.
C  This routine also uses the COMMON variables EL0, H, TN, UROUND,
C  MITER, N, NFE, and NJE.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  DGBFA, DGEFA, DVNORM
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890504  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DPREPJ
C**End
      EXTERNAL F, JAC
      INTEGER NEQ, NYH, IWM
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     2   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, I1, I2, IER, II, J, J1, JJ, LENP,
     1   MBA, MBAND, MEB1, MEBAND, ML, ML3, MU, NP1
      DOUBLE PRECISION Y, YH, EWT, FTEM, SAVF, WM
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION CON, DI, FAC, HL0, R, R0, SRUR, YI, YJ, YJJ,
     1   DVNORM
      DIMENSION NEQ(*), Y(*), YH(NYH,*), EWT(*), FTEM(*), SAVF(*),
     1   WM(*), IWM(*)
      COMMON /DLS001/ ROWNS(209),
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     3   IOWND(12), IOWNS(6),
     4   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
C***FIRST EXECUTABLE STATEMENT  DPREPJ
      NJE = NJE + 1
      IERPJ = 0
      JCUR = 1
      HL0 = H*EL0
      GO TO (100, 200, 300, 400, 500), MITER
C If MITER = 1, call JAC and multiply by scalar. -----------------------
 100  LENP = N*N
      DO 110 I = 1,LENP
 110    WM(I+2) = 0.0D0
      CALL JAC (NEQ, TN, Y, 0, 0, WM(3), N)
      CON = -HL0
      DO 120 I = 1,LENP
 120    WM(I+2) = WM(I+2)*CON
      GO TO 240
C If MITER = 2, make N calls to F to approximate J. --------------------
 200  FAC = DVNORM (N, SAVF, EWT)
      R0 = 1000.0D0*ABS(H)*UROUND*N*FAC
      IF (R0 .EQ. 0.0D0) R0 = 1.0D0
      SRUR = WM(1)
      J1 = 2
      DO 230 J = 1,N
        YJ = Y(J)
        R = MAX(SRUR*ABS(YJ),R0/EWT(J))
```

```
        Y(J) = Y(J) + R
        FAC = -HL0/R
        CALL F (NEQ, TN, Y, FTEM)
        DO 220 I = 1,N
 220      WM(I+J1) = (FTEM(I) - SAVF(I))*FAC
        Y(J) = YJ
        J1 = J1 + N
 230    CONTINUE
      NFE = NFE + N
C Add identity matrix. -------------------------------------------------
 240  J = 3
      NP1 = N + 1
      DO 250 I = 1,N
        WM(J) = WM(J) + 1.0D0
 250    J = J + NP1
C Do LU decomposition on P. --------------------------------------------
      CALL DGEFA (WM(3), N, N, IWM(21), IER)
      IF (IER .NE. 0) IERPJ = 1
      RETURN
C If MITER = 3, construct a diagonal approximation to J and P. ---------
 300  WM(2) = HL0
      R = EL0*0.1D0
      DO 310 I = 1,N
 310    Y(I) = Y(I) + R*(H*SAVF(I) - YH(I,2))
      CALL F (NEQ, TN, Y, WM(3))
      NFE = NFE + 1
      DO 320 I = 1,N
        R0 = H*SAVF(I) - YH(I,2)
        DI = 0.1D0*R0 - H*(WM(I+2) - SAVF(I))
        WM(I+2) = 1.0D0
        IF (ABS(R0) .LT. UROUND/EWT(I)) GO TO 320
        IF (ABS(DI) .EQ. 0.0D0) GO TO 330
        WM(I+2) = 0.1D0*R0/DI
 320    CONTINUE
      RETURN
 330  IERPJ = 1
      RETURN
C If MITER = 4, call JAC and multiply by scalar. -----------------------
 400  ML = IWM(1)
      MU = IWM(2)
      ML3 = ML + 3
      MBAND = ML + MU + 1
      MEBAND = MBAND + ML
      LENP = MEBAND*N
      DO 410 I = 1,LENP
 410    WM(I+2) = 0.0D0
      CALL JAC (NEQ, TN, Y, ML, MU, WM(ML3), MEBAND)
      CON = -HL0
      DO 420 I = 1,LENP
 420    WM(I+2) = WM(I+2)*CON
      GO TO 570
C If MITER = 5, make MBAND calls to F to approximate J. ----------------
 500  ML = IWM(1)
      MU = IWM(2)
      MBAND = ML + MU + 1
      MBA = MIN(MBAND,N)
      MEBAND = MBAND + ML
      MEB1 = MEBAND - 1
      SRUR = WM(1)
      FAC = DVNORM (N, SAVF, EWT)
      R0 = 1000.0D0*ABS(H)*UROUND*N*FAC
      IF (R0 .EQ. 0.0D0) R0 = 1.0D0
      DO 560 J = 1,MBA
        DO 530 I = J,N,MBAND
          YI = Y(I)
          R = MAX(SRUR*ABS(YI),R0/EWT(I))
 530      Y(I) = Y(I) + R
```

```
        CALL F (NEQ, TN, Y, FTEM)
        DO 550 JJ = J,N,MBAND
          Y(JJ) = YH(JJ,1)
          YJJ = Y(JJ)
          R = MAX(SRUR*ABS(YJJ),R0/EWT(JJ))
          FAC = -HL0/R
          I1 = MAX(JJ-MU,1)
          I2 = MIN(JJ+ML,N)
          II = JJ*MEB1 - ML + 2
          DO 540 I = I1,I2
 540        WM(II+I) = (FTEM(I) - SAVF(I))*FAC
 550      CONTINUE
 560    CONTINUE
      NFE = NFE + MBA
C Add identity matrix. -------------------------------------------------
 570  II = MBAND + 2
      DO 580 I = 1,N
        WM(II) = WM(II) + 1.0D0
 580    II = II + MEBAND
C Do LU decomposition of P. --------------------------------------------
      CALL DGBFA (WM(3), MEBAND, N, ML, MU, IWM(21), IER)
      IF (IER .NE. 0) IERPJ = 1
      RETURN
C----------------------- END OF SUBROUTINE DPREPJ ----------------------
      END
*DECK DSOLSY
      SUBROUTINE DSOLSY (WM, IWM, X, TEM)
C***BEGIN PROLOGUE  DSOLSY
C***SUBSIDIARY
C***PURPOSE  ODEPACK linear system solver.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SSOLSY-S, DSOLSY-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This routine manages the solution of the linear system arising from
C  a chord iteration.  It is called if MITER .ne. 0.
C  If MITER is 1 or 2, it calls DGESL to accomplish this.
C  If MITER = 3 it updates the coefficient h*EL0 in the diagonal
C  matrix, and then computes the solution.
C  If MITER is 4 or 5, it calls DGBSL.
C  Communication with DSOLSY uses the following variables:
C  WM    = real work space containing the inverse diagonal matrix if
C          MITER = 3 and the LU decomposition of the matrix otherwise.
C          Storage of matrix elements starts at WM(3).
C          WM also contains the following matrix-related data:
C          WM(1) = SQRT(UROUND) (not used here),
C          WM(2) = HL0, the previous value of h*EL0, used if MITER = 3.
C  IWM   = integer work space containing pivot information, starting at
C          IWM(21), if MITER is 1, 2, 4, or 5.  IWM also contains band
C          parameters ML = IWM(1) and MU = IWM(2) if MITER is 4 or 5.
C  X     = the right-hand side vector on input, and the solution vector
C          on output, of length N.
C  TEM   = vector of work space of length N, not used in this version.
C  IERSL = output flag (in COMMON).  IERSL = 0 if no trouble occurred.
C          IERSL = 1 if a singular matrix arose with MITER = 3.
C  This routine also uses the COMMON variables EL0, H, MITER, and N.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  DGBSL, DGESL
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DSOLSY
```

```
C**End
      INTEGER IWM
      INTEGER IOWND, IOWNS,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     2   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
      INTEGER I, MEBAND, ML, MU
      DOUBLE PRECISION WM, X, TEM
      DOUBLE PRECISION ROWNS,
     1   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION DI, HL0, PHL0, R
      DIMENSION WM(*), IWM(*), X(*), TEM(*)
      COMMON /DLS001/ ROWNS(209),
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND,
     3   IOWND(12), IOWNS(6),
     4   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
C***FIRST EXECUTABLE STATEMENT  DSOLSY
      IERSL = 0
      GO TO (100, 100, 300, 400, 400), MITER
 100  CALL DGESL (WM(3), N, N, IWM(21), X, 0)
      RETURN
C
 300  PHL0 = WM(2)
      HL0 = H*EL0
      WM(2) = HL0
      IF (HL0 .EQ. PHL0) GO TO 330
      R = HL0/PHL0
      DO 320 I = 1,N
        DI = 1.0D0 - R*(1.0D0 - 1.0D0/WM(I+2))
        IF (ABS(DI) .EQ. 0.0D0) GO TO 390
 320    WM(I+2) = 1.0D0/DI
 330  DO 340 I = 1,N
 340    X(I) = WM(I+2)*X(I)
      RETURN
 390  IERSL = 1
      RETURN
C
 400  ML = IWM(1)
      MU = IWM(2)
      MEBAND = 2*ML + MU + 1
      CALL DGBSL (WM(3), MEBAND, N, ML, MU, IWM(21), X, 0)
      RETURN
C----------------------- END OF SUBROUTINE DSOLSY ----------------------
      END
*DECK DSRCOM
      SUBROUTINE DSRCOM (RSAV, ISAV, JOB)
C***BEGIN PROLOGUE  DSRCOM
C***SUBSIDIARY
C***PURPOSE  Save/restore ODEPACK COMMON blocks.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SSRCOM-S, DSRCOM-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This routine saves or restores (depending on JOB) the contents of
C  the COMMON block DLS001, which is used internally
C  by one or more ODEPACK solvers.
C
C  RSAV = real array of length 218 or more.
C  ISAV = integer array of length 37 or more.
C  JOB  = flag indicating to save or restore the COMMON blocks:
C         JOB  = 1 if COMMON is to be saved (written to RSAV/ISAV)
C         JOB  = 2 if COMMON is to be restored (read from RSAV/ISAV)
C         A call with JOB = 2 presumes a prior call with JOB = 1.
C
C***SEE ALSO  DLSODE
```

```
C***ROUTINES CALLED  (NONE)
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   921116  Deleted treatment of block /EH0001/.  (ACH)
C   930801  Reduced Common block length by 2.  (ACH)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DSRCOM
C**End
      INTEGER ISAV, JOB
      INTEGER ILS
      INTEGER I, LENILS, LENRLS
      DOUBLE PRECISION RSAV,   RLS
      DIMENSION RSAV(*), ISAV(*)
      COMMON /DLS001/ RLS(218), ILS(37)
      DATA LENRLS/218/, LENILS/37/
C
C***FIRST EXECUTABLE STATEMENT  DSRCOM
      IF (JOB .EQ. 2) GO TO 100
C
      DO 10 I = 1,LENRLS
 10     RSAV(I) = RLS(I)
      DO 20 I = 1,LENILS
 20     ISAV(I) = ILS(I)
      RETURN
C
 100  CONTINUE
      DO 110 I = 1,LENRLS
 110     RLS(I) = RSAV(I)
      DO 120 I = 1,LENILS
 120     ILS(I) = ISAV(I)
      RETURN
C----------------------- END OF SUBROUTINE DSRCOM ----------------------
      END
*DECK DSTODE
      SUBROUTINE DSTODE (NEQ, Y, YH, NYH, YH1, EWT, SAVF, ACOR,
     1   WM, IWM, F, JAC, PJAC, SLVS)
C***BEGIN PROLOGUE  DSTODE
C***SUBSIDIARY
C***PURPOSE  Performs one step of an ODEPACK integration.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SSTODE-S, DSTODE-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  DSTODE performs one step of the integration of an initial value
C  problem for a system of ordinary differential equations.
C  Note:  DSTODE is independent of the value of the iteration method
C  indicator MITER, when this is .ne. 0, and hence is independent
C  of the type of chord method used, or the Jacobian structure.
C  Communication with DSTODE is done with the following variables:
C
C  NEQ    = integer array containing problem size in NEQ(1), and
C           passed as the NEQ argument in all calls to F and JAC.
C  Y      = an array of length .ge. N used as the Y argument in
C           all calls to F and JAC.
C  YH     = an NYH by LMAX array containing the dependent variables
C           and their approximate scaled derivatives, where
C           LMAX = MAXORD + 1.  YH(i,j+1) contains the approximate
C           j-th derivative of y(i), scaled by h**j/factorial(j)
C           (j = 0,1,...,NQ).  on entry for the first step, the first
C           two columns of YH must be set from the initial values.
C  NYH    = a constant integer .ge. N, the first dimension of YH.
C  YH1    = a one-dimensional array occupying the same space as YH.
C  EWT    = an array of length N containing multiplicative weights
```

```
C           for local error measurements.  Local errors in Y(i) are
C           compared to 1.0/EWT(i) in various error tests.
C  SAVF   = an array of working storage, of length N.
C           Also used for input of YH(*,MAXORD+2) when JSTART = -1
C           and MAXORD .lt. the current order NQ.
C  ACOR   = a work array of length N, used for the accumulated
C           corrections.  On a successful return, ACOR(i) contains
C           the estimated one-step local error in Y(i).
C  WM,IWM = real and integer work arrays associated with matrix
C           operations in chord iteration (MITER .ne. 0).
C  PJAC   = name of routine to evaluate and preprocess Jacobian matrix
C           and P = I - h*el0*JAC, if a chord method is being used.
C  SLVS   = name of routine to solve linear system in chord iteration.
C  CCMAX  = maximum relative change in h*el0 before PJAC is called.
C  H      = the step size to be attempted on the next step.
C           H is altered by the error control algorithm during the
C           problem.  H can be either positive or negative, but its
C           sign must remain constant throughout the problem.
C  HMIN   = the minimum absolute value of the step size h to be used.
C  HMXI   = inverse of the maximum absolute value of h to be used.
C           HMXI = 0.0 is allowed and corresponds to an infinite hmax.
C           HMIN and HMXI may be changed at any time, but will not
C           take effect until the next change of h is considered.
C  TN     = the independent variable. TN is updated on each step taken.
C  JSTART = an integer used for input only, with the following
C           values and meanings:
C                0  perform the first step.
C            .gt.0  take a new step continuing from the last.
C               -1  take the next step with a new value of H, MAXORD,
C                     N, METH, MITER, and/or matrix parameters.
C               -2  take the next step with a new value of H,
C                     but with other inputs unchanged.
C           On return, JSTART is set to 1 to facilitate continuation.
C  KFLAG  = a completion code with the following meanings:
C                0  the step was succesful.
C               -1  the requested error could not be achieved.
C               -2  corrector convergence could not be achieved.
C               -3  fatal error in PJAC or SLVS.
C           A return with KFLAG = -1 or -2 means either
C           abs(H) = HMIN or 10 consecutive failures occurred.
C           On a return with KFLAG negative, the values of TN and
C           the YH array are as of the beginning of the last
C           step, and H is the last step size attempted.
C  MAXORD = the maximum order of integration method to be allowed.
C  MAXCOR = the maximum number of corrector iterations allowed.
C  MSBP   = maximum number of steps between PJAC calls (MITER .gt. 0).
C  MXNCF  = maximum number of convergence failures allowed.
C  METH/MITER = the method flags.  See description in driver.
C  N      = the number of first-order differential equations.
C  The values of CCMAX, H, HMIN, HMXI, TN, JSTART, KFLAG, MAXORD,
C  MAXCOR, MSBP, MXNCF, METH, MITER, and N are communicated via COMMON.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  DCFODE, DVNORM
C***COMMON BLOCKS    DLS001
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DSTODE
C**End
      EXTERNAL F, JAC, PJAC, SLVS
      INTEGER NEQ, NYH, IWM
      INTEGER IOWND, IALTH, IPUP, LMAX, MEO, NQNYH, NSLP,
     1   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     2   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
```

```
      INTEGER I, I1, IREDO, IRET, J, JB, M, NCF, NEWQ
      DOUBLE PRECISION Y, YH, YH1, EWT, SAVF, ACOR, WM
      DOUBLE PRECISION CONIT, CRATE, EL, ELCO, HOLD, RMAX, TESCO,
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND
      DOUBLE PRECISION DCON, DDN, DEL, DELP, DSM, DUP, EXDN, EXSM, EXUP,
     1   R, RH, RHDN, RHSM, RHUP, TOLD, DVNORM
      DIMENSION NEQ(*), Y(*), YH(NYH,*), YH1(*), EWT(*), SAVF(*),
     1   ACOR(*), WM(*), IWM(*)
      COMMON /DLS001/ CONIT, CRATE, EL(13), ELCO(13,12),
     1   HOLD, RMAX, TESCO(3,12),
     2   CCMAX, EL0, H, HMIN, HMXI, HU, RC, TN, UROUND, IOWND(12),
     3   IALTH, IPUP, LMAX, MEO, NQNYH, NSLP,
     4   ICF, IERPJ, IERSL, JCUR, JSTART, KFLAG, L, METH, MITER,
     5   MAXORD, MAXCOR, MSBP, MXNCF, N, NQ, NST, NFE, NJE, NQU
C
C***FIRST EXECUTABLE STATEMENT  DSTODE
      KFLAG = 0
      TOLD = TN
      NCF = 0
      IERPJ = 0
      IERSL = 0
      JCUR = 0
      ICF = 0
      DELP = 0.0D0
      IF (JSTART .GT. 0) GO TO 200
      IF (JSTART .EQ. -1) GO TO 100
      IF (JSTART .EQ. -2) GO TO 160
C-----------------------------------------------------------------------
C On the first call, the order is set to 1, and other variables are
C initialized.  RMAX is the maximum ratio by which H can be increased
C in a single step.  It is initially 1.E4 to compensate for the small
C initial H, but then is normally equal to 10.  If a failure
C occurs (in corrector convergence or error test), RMAX is set to 2
C for the next increase.
C-----------------------------------------------------------------------
      LMAX = MAXORD + 1
      NQ = 1
      L = 2
      IALTH = 2
      RMAX = 10000.0D0
      RC = 0.0D0
      EL0 = 1.0D0
      CRATE = 0.7D0
      HOLD = H
      MEO = METH
      NSLP = 0
      IPUP = MITER
      IRET = 3
      GO TO 140
C-----------------------------------------------------------------------
C The following block handles preliminaries needed when JSTART = -1.
C IPUP is set to MITER to force a matrix update.
C If an order increase is about to be considered (IALTH = 1),
C IALTH is reset to 2 to postpone consideration one more step.
C If the caller has changed METH, DCFODE is called to reset
C the coefficients of the method.
C If the caller has changed MAXORD to a value less than the current
C order NQ, NQ is reduced to MAXORD, and a new H chosen accordingly.
C If H is to be changed, YH must be rescaled.
C If H or METH is being changed, IALTH is reset to L = NQ + 1
C to prevent further changes in H for that many steps.
C-----------------------------------------------------------------------
 100  IPUP = MITER
      LMAX = MAXORD + 1
      IF (IALTH .EQ. 1) IALTH = 2
      IF (METH .EQ. MEO) GO TO 110
      CALL DCFODE (METH, ELCO, TESCO)
```

```
      MEO = METH
      IF (NQ .GT. MAXORD) GO TO 120
      IALTH = L
      IRET = 1
      GO TO 150
 110  IF (NQ .LE. MAXORD) GO TO 160
 120  NQ = MAXORD
      L = LMAX
      DO 125 I = 1,L
 125    EL(I) = ELCO(I,NQ)
      NQNYH = NQ*NYH
      RC = RC*EL(1)/EL0
      EL0 = EL(1)
      CONIT = 0.5D0/(NQ+2)
      DDN = DVNORM (N, SAVF, EWT)/TESCO(1,L)
      EXDN = 1.0D0/L
      RHDN = 1.0D0/(1.3D0*DDN**EXDN + 0.0000013D0)
      RH = MIN(RHDN,1.0D0)
      IREDO = 3
      IF (H .EQ. HOLD) GO TO 170
      RH = MIN(RH,ABS(H/HOLD))
      H = HOLD
      GO TO 175
C-----------------------------------------------------------------------
C DCFODE is called to get all the integration coefficients for the
C current METH.  Then the EL vector and related constants are reset
C whenever the order NQ is changed, or at the start of the problem.
C-----------------------------------------------------------------------
 140  CALL DCFODE (METH, ELCO, TESCO)
 150  DO 155 I = 1,L
 155    EL(I) = ELCO(I,NQ)
      NQNYH = NQ*NYH
      RC = RC*EL(1)/EL0
      EL0 = EL(1)
      CONIT = 0.5D0/(NQ+2)
      GO TO (160, 170, 200), IRET
C-----------------------------------------------------------------------
C If H is being changed, the H ratio RH is checked against
C RMAX, HMIN, and HMXI, and the YH array rescaled.  IALTH is set to
C L = NQ + 1 to prevent a change of H for that many steps, unless
C forced by a convergence or error test failure.
C-----------------------------------------------------------------------
 160  IF (H .EQ. HOLD) GO TO 200
      RH = H/HOLD
      H = HOLD
      IREDO = 3
      GO TO 175
 170  RH = MAX(RH,HMIN/ABS(H))
 175  RH = MIN(RH,RMAX)
      RH = RH/MAX(1.0D0,ABS(H)*HMXI*RH)
      R = 1.0D0
      DO 180 J = 2,L
        R = R*RH
        DO 180 I = 1,N
 180      YH(I,J) = YH(I,J)*R
      H = H*RH
      RC = RC*RH
      IALTH = L
      IF (IREDO .EQ. 0) GO TO 690
C-----------------------------------------------------------------------
C This section computes the predicted values by effectively
C multiplying the YH array by the Pascal Triangle matrix.
C RC is the ratio of new to old values of the coefficient  H*EL(1).
C When RC differs from 1 by more than CCMAX, IPUP is set to MITER
C to force PJAC to be called, if a Jacobian is involved.
C In any case, PJAC is called at least every MSBP steps.
C-----------------------------------------------------------------------
```

```
 200  IF (ABS(RC-1.0D0) .GT. CCMAX) IPUP = MITER
      IF (NST .GE. NSLP+MSBP) IPUP = MITER
      TN = TN + H
      I1 = NQNYH + 1
      DO 215 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 210 I = I1,NQNYH
 210      YH1(I) = YH1(I) + YH1(I+NYH)
 215    CONTINUE
C-----------------------------------------------------------------------
C Up to MAXCOR corrector iterations are taken.  A convergence test is
C made on the R.M.S. norm of each correction, weighted by the error
C weight vector EWT.  The sum of the corrections is accumulated in the
C vector ACOR(i).  The YH array is not altered in the corrector loop.
C-----------------------------------------------------------------------
 220  M = 0
      DO 230 I = 1,N
 230    Y(I) = YH(I,1)
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      IF (IPUP .LE. 0) GO TO 250
C-----------------------------------------------------------------------
C If indicated, the matrix P = I - h*el(1)*J is reevaluated and
C preprocessed before starting the corrector iteration.  IPUP is set
C to 0 as an indicator that this has been done.
C-----------------------------------------------------------------------
      CALL PJAC (NEQ, Y, YH, NYH, EWT, ACOR, SAVF, WM, IWM, F, JAC)
      IPUP = 0
      RC = 1.0D0
      NSLP = NST
      CRATE = 0.7D0
      IF (IERPJ .NE. 0) GO TO 430
 250  DO 260 I = 1,N
 260    ACOR(I) = 0.0D0
 270  IF (MITER .NE. 0) GO TO 350
C-----------------------------------------------------------------------
C In the case of functional iteration, update Y directly from
C the result of the last function evaluation.
C-----------------------------------------------------------------------
      DO 290 I = 1,N
        SAVF(I) = H*SAVF(I) - YH(I,2)
 290    Y(I) = SAVF(I) - ACOR(I)
      DEL = DVNORM (N, Y, EWT)
      DO 300 I = 1,N
        Y(I) = YH(I,1) + EL(1)*SAVF(I)
 300    ACOR(I) = SAVF(I)
      GO TO 400
C-----------------------------------------------------------------------
C In the case of the chord method, compute the corrector error,
C and solve the linear system with that as right-hand side and
C P as coefficient matrix.
C-----------------------------------------------------------------------
 350  DO 360 I = 1,N
 360    Y(I) = H*SAVF(I) - (YH(I,2) + ACOR(I))
      CALL SLVS (WM, IWM, Y, SAVF)
      IF (IERSL .LT. 0) GO TO 430
      IF (IERSL .GT. 0) GO TO 410
      DEL = DVNORM (N, Y, EWT)
      DO 380 I = 1,N
        ACOR(I) = ACOR(I) + Y(I)
 380    Y(I) = YH(I,1) + EL(1)*ACOR(I)
C-----------------------------------------------------------------------
C Test for convergence.  If M.gt.0, an estimate of the convergence
C rate constant is stored in CRATE, and this is used in the test.
C-----------------------------------------------------------------------
 400  IF (M .NE. 0) CRATE = MAX(0.2D0*CRATE,DEL/DELP)
```

```
      DCON = DEL*MIN(1.0D0,1.5D0*CRATE)/(TESCO(2,NQ)*CONIT)
      IF (DCON .LE. 1.0D0) GO TO 450
      M = M + 1
      IF (M .EQ. MAXCOR) GO TO 410
      IF (M .GE. 2 .AND. DEL .GT. 2.0D0*DELP) GO TO 410
      DELP = DEL
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      GO TO 270
C-----------------------------------------------------------------------
C The corrector iteration failed to converge.
C If MITER .ne. 0 and the Jacobian is out of date, PJAC is called for
C the next try.  Otherwise the YH array is retracted to its values
C before prediction, and H is reduced, if possible.  If H cannot be
C reduced or MXNCF failures have occurred, exit with KFLAG = -2.
C-----------------------------------------------------------------------
 410  IF (MITER .EQ. 0 .OR. JCUR .EQ. 1) GO TO 430
      ICF = 1
      IPUP = MITER
      GO TO 220
 430  ICF = 2
      NCF = NCF + 1
      RMAX = 2.0D0
      TN = TOLD
      I1 = NQNYH + 1
      DO 445 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 440 I = I1,NQNYH
 440      YH1(I) = YH1(I) - YH1(I+NYH)
 445    CONTINUE
      IF (IERPJ .LT. 0 .OR. IERSL .LT. 0) GO TO 680
      IF (ABS(H) .LE. HMIN*1.00001D0) GO TO 670
      IF (NCF .EQ. MXNCF) GO TO 670
      RH = 0.25D0
      IPUP = MITER
      IREDO = 1
      GO TO 170
C-----------------------------------------------------------------------
C The corrector has converged.  JCUR is set to 0
C to signal that the Jacobian involved may need updating later.
C The local error test is made and control passes to statement 500
C if it fails.
C-----------------------------------------------------------------------
 450  JCUR = 0
      IF (M .EQ. 0) DSM = DEL/TESCO(2,NQ)
      IF (M .GT. 0) DSM = DVNORM (N, ACOR, EWT)/TESCO(2,NQ)
      IF (DSM .GT. 1.0D0) GO TO 500
C-----------------------------------------------------------------------
C After a successful step, update the YH array.
C Consider changing H if IALTH = 1.  Otherwise decrease IALTH by 1.
C If IALTH is then 1 and NQ .lt. MAXORD, then ACOR is saved for
C use in a possible order increase on the next step.
C If a change in H is considered, an increase or decrease in order
C by one is considered also.  A change in H is made only if it is by a
C factor of at least 1.1.  If not, IALTH is set to 3 to prevent
C testing for that many steps.
C-----------------------------------------------------------------------
      KFLAG = 0
      IREDO = 0
      NST = NST + 1
      HU = H
      NQU = NQ
      DO 470 J = 1,L
        DO 470 I = 1,N
 470      YH(I,J) = YH(I,J) + EL(J)*ACOR(I)
      IALTH = IALTH - 1
```

```
      IF (IALTH .EQ. 0) GO TO 520
      IF (IALTH .GT. 1) GO TO 700
      IF (L .EQ. LMAX) GO TO 700
      DO 490 I = 1,N
 490    YH(I,LMAX) = ACOR(I)
      GO TO 700
C-----------------------------------------------------------------------
C The error test failed.  KFLAG keeps track of multiple failures.
C Restore TN and the YH array to their previous values, and prepare
C to try the step again.  Compute the optimum step size for this or
C one lower order.  After 2 or more failures, H is forced to decrease
C by a factor of 0.2 or less.
C-----------------------------------------------------------------------
 500  KFLAG = KFLAG - 1
      TN = TOLD
      I1 = NQNYH + 1
      DO 515 JB = 1,NQ
        I1 = I1 - NYH
Cdir$ ivdep
        DO 510 I = I1,NQNYH
 510      YH1(I) = YH1(I) - YH1(I+NYH)
 515    CONTINUE
      RMAX = 2.0D0
      IF (ABS(H) .LE. HMIN*1.00001D0) GO TO 660
      IF (KFLAG .LE. -3) GO TO 640
      IREDO = 2
      RHUP = 0.0D0
      GO TO 540
C-----------------------------------------------------------------------
C Regardless of the success or failure of the step, factors
C RHDN, RHSM, and RHUP are computed, by which H could be multiplied
C at order NQ - 1, order NQ, or order NQ + 1, respectively.
C In the case of failure, RHUP = 0.0 to avoid an order increase.
C The largest of these is determined and the new order chosen
C accordingly.  If the order is to be increased, we compute one
C additional scaled derivative.
C-----------------------------------------------------------------------
 520  RHUP = 0.0D0
      IF (L .EQ. LMAX) GO TO 540
      DO 530 I = 1,N
 530    SAVF(I) = ACOR(I) - YH(I,LMAX)
      DUP = DVNORM (N, SAVF, EWT)/TESCO(3,NQ)
      EXUP = 1.0D0/(L+1)
      RHUP = 1.0D0/(1.4D0*DUP**EXUP + 0.0000014D0)
 540  EXSM = 1.0D0/L
      RHSM = 1.0D0/(1.2D0*DSM**EXSM + 0.0000012D0)
      RHDN = 0.0D0
      IF (NQ .EQ. 1) GO TO 560
      DDN = DVNORM (N, YH(1,L), EWT)/TESCO(1,NQ)
      EXDN = 1.0D0/NQ
      RHDN = 1.0D0/(1.3D0*DDN**EXDN + 0.0000013D0)
 560  IF (RHSM .GE. RHUP) GO TO 570
      IF (RHUP .GT. RHDN) GO TO 590
      GO TO 580
 570  IF (RHSM .LT. RHDN) GO TO 580
      NEWQ = NQ
      RH = RHSM
      GO TO 620
 580  NEWQ = NQ - 1
      RH = RHDN
      IF (KFLAG .LT. 0 .AND. RH .GT. 1.0D0) RH = 1.0D0
      GO TO 620
 590  NEWQ = L
      RH = RHUP
      IF (RH .LT. 1.1D0) GO TO 610
      R = EL(L)/L
      DO 600 I = 1,N
```

```
 600    YH(I,NEWQ+1) = ACOR(I)*R
      GO TO 630
 610  IALTH = 3
      GO TO 700
 620  IF ((KFLAG .EQ. 0) .AND. (RH .LT. 1.1D0)) GO TO 610
      IF (KFLAG .LE. -2) RH = MIN(RH,0.2D0)
C-----------------------------------------------------------------------
C If there is a change of order, reset NQ, l, and the coefficients.
C In any case H is reset according to RH and the YH array is rescaled.
C Then exit from 690 if the step was OK, or redo the step otherwise.
C-----------------------------------------------------------------------
      IF (NEWQ .EQ. NQ) GO TO 170
 630  NQ = NEWQ
      L = NQ + 1
      IRET = 2
      GO TO 150
C-----------------------------------------------------------------------
C Control reaches this section if 3 or more failures have occured.
C If 10 failures have occurred, exit with KFLAG = -1.
C It is assumed that the derivatives that have accumulated in the
C YH array have errors of the wrong order.  Hence the first
C derivative is recomputed, and the order is set to 1.  Then
C H is reduced by a factor of 10, and the step is retried,
C until it succeeds or H reaches HMIN.
C-----------------------------------------------------------------------
 640  IF (KFLAG .EQ. -10) GO TO 660
      RH = 0.1D0
      RH = MAX(HMIN/ABS(H),RH)
      H = H*RH
      DO 645 I = 1,N
 645    Y(I) = YH(I,1)
      CALL F (NEQ, TN, Y, SAVF)
      NFE = NFE + 1
      DO 650 I = 1,N
 650    YH(I,2) = H*SAVF(I)
      IPUP = MITER
      IALTH = 5
      IF (NQ .EQ. 1) GO TO 200
      NQ = 1
      L = 2
      IRET = 3
      GO TO 150
C-----------------------------------------------------------------------
C All returns are made through this section.  H is saved in HOLD
C to allow the caller to change H on the next step.
C-----------------------------------------------------------------------
 660  KFLAG = -1
      GO TO 720
 670  KFLAG = -2
      GO TO 720
 680  KFLAG = -3
      GO TO 720
 690  RMAX = 10.0D0
 700  R = 1.0D0/TESCO(2,NQU)
      DO 710 I = 1,N
 710    ACOR(I) = ACOR(I)*R
 720  HOLD = H
      JSTART = 1
      RETURN
C----------------------- END OF SUBROUTINE DSTODE ----------------------
      END
*DECK DEWSET
      SUBROUTINE DEWSET (N, ITOL, RTOL, ATOL, YCUR, EWT)
C***BEGIN PROLOGUE  DEWSET
C***SUBSIDIARY
C***PURPOSE  Set error weight vector.
C***LIBRARY   MATHLIB (ODEPACK)
```

```
C***TYPE      DOUBLE PRECISION (SEWSET-S, DEWSET-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This subroutine sets the error weight vector EWT according to
C      EWT(i) = RTOL(i)*ABS(YCUR(i)) + ATOL(i),  i = 1,...,N,
C  with the subscript on RTOL and/or ATOL possibly replaced by 1 above,
C  depending on the value of ITOL.
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DEWSET
C**End
      INTEGER N, ITOL
      INTEGER I
      DOUBLE PRECISION RTOL, ATOL, YCUR, EWT
      DIMENSION RTOL(*), ATOL(*), YCUR(N), EWT(N)
C
C***FIRST EXECUTABLE STATEMENT  DEWSET
      GO TO (10, 20, 30, 40), ITOL
 10   CONTINUE
      DO 15 I = 1,N
 15     EWT(I) = RTOL(1)*ABS(YCUR(I)) + ATOL(1)
      RETURN
 20   CONTINUE
      DO 25 I = 1,N
 25     EWT(I) = RTOL(1)*ABS(YCUR(I)) + ATOL(I)
      RETURN
 30   CONTINUE
      DO 35 I = 1,N
 35     EWT(I) = RTOL(I)*ABS(YCUR(I)) + ATOL(1)
      RETURN
 40   CONTINUE
      DO 45 I = 1,N
 45     EWT(I) = RTOL(I)*ABS(YCUR(I)) + ATOL(I)
      RETURN
C----------------------- END OF SUBROUTINE DEWSET ------------------------
      END
*DECK DVNORM
      DOUBLE PRECISION FUNCTION DVNORM (N, V, W)
C***BEGIN PROLOGUE  DVNORM
C***SUBSIDIARY
C***PURPOSE  Weighted root-mean-square vector norm.
C***LIBRARY   MATHLIB (ODEPACK)
C***TYPE      DOUBLE PRECISION (SVNORM-S, DVNORM-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  This function routine computes the weighted root-mean-square norm
C  of the vector of length N contained in the array V, with weights
C  contained in the array W of length N:
C    DVNORM = SQRT( (1/N) * SUM( V(i)*W(i) )**2 )
C
C***SEE ALSO  DLSODE
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   791129  DATE WRITTEN
C   890501  Modified prologue to SLATEC/LDOC format.  (FNF)
C   890503  Minor cosmetic changes.  (FNF)
C   930809  Renamed to allow single/double precision versions. (ACH)
C***END PROLOGUE  DVNORM
C**End
```

```
      INTEGER N,   I
      DOUBLE PRECISION V, W,   SUM
      DIMENSION V(N), W(N)
C
C***FIRST EXECUTABLE STATEMENT  DVNORM
      SUM = 0.0D0
      DO 10 I = 1,N
 10     SUM = SUM + (V(I)*W(I))**2
      DVNORM = SQRT(SUM/N)
      RETURN
C----------------------- END OF FUNCTION DVNORM -----------------------
      END
*DECK DUMACH
      DOUBLE PRECISION FUNCTION DUMACH ()
C***BEGIN PROLOGUE  DUMACH
C***PURPOSE  Compute the unit roundoff of the machine.
C***LIBRARY   MATHLIB
C***CATEGORY  R1
C***TYPE      DOUBLE PRECISION (RUMACH-S, DUMACH-D)
C***KEYWORDS  MACHINE CONSTANTS
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C *Usage:
C        DOUBLE PRECISION  A, DUMACH
C        A = DUMACH()
C
C *Function Return Values:
C     A : the unit roundoff of the machine.
C
C *Description:
C     The unit roundoff is defined as the smallest positive machine
C     number u such that  1.0 + u .ne. 1.0.  This is computed by DUMACH
C     in a machine-independent manner.
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   930216  DATE WRITTEN
C   930818  Added SLATEC-format prologue.  (FNF)
C***END PROLOGUE  DUMACH
C
C*Internal Notes:
C-----------------------------------------------------------------------
C Subroutines/functions called by DUMACH.. None
C-----------------------------------------------------------------------
C**End
C
      DOUBLE PRECISION U, COMP
C***FIRST EXECUTABLE STATEMENT  DUMACH
      U = 1.0D0
 10   U = U*0.5D0
      COMP = 1.0D0 + U
      IF (COMP .NE. 1.0D0) GO TO 10
      DUMACH = U*2.0D0
      RETURN
C----------------------- End of Function DUMACH -----------------------
      END
*DECK XERRWD
      SUBROUTINE XERRWD (MSG, NMES, NERR, LEVEL, NI, I1, I2, NR, R1, R2)
C***BEGIN PROLOGUE  XERRWD
C***SUBSIDIARY
C***PURPOSE  Write error message with values.
C***LIBRARY   MATHLIB
C***CATEGORY  R3C
C***TYPE      DOUBLE PRECISION (XERRWV-S, XERRWD-D)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
```

```
C
C  Subroutines XERRWD, XSETF, XSETUN, and the function routine IXSAV,
C  as given here, constitute a simplified version of the SLATEC error
C  handling package.
C
C  All arguments are input arguments.
C
C  MSG    = The message (character array).
C  NMES   = The length of MSG (number of characters).
C  NERR   = The error number (not used).
C  LEVEL  = The error level..
C           0 or 1 means recoverable (control returns to caller).
C           2 means fatal (run is aborted--see note below).
C  NI     = Number of integers (0, 1, or 2) to be printed with message.
C  I1,I2  = Integers to be printed, depending on NI.
C  NR     = Number of reals (0, 1, or 2) to be printed with message.
C  R1,R2  = Reals to be printed, depending on NR.
C
C  Note..  this routine is machine-dependent and specialized for use
C  in limited context, in the following ways..
C  1. The argument MSG is assumed to be of type CHARACTER, and
C     the message is printed with a format of (1X,A).
C  2. The message is assumed to take only one line.
C     Multi-line messages are generated by repeated calls.
C  3. If LEVEL = 2, control passes to the statement   STOP
C     to abort the run.  This statement may be machine-dependent.
C  4. R1 and R2 are assumed to be in double precision and are printed
C     in D21.13 format.
C
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   920831  DATE WRITTEN
C   921118  Replaced MFLGSV/LUNSAV by IXSAV. (ACH)
C   930329  Modified prologue to SLATEC format. (FNF)
C   930407  Changed MSG from CHARACTER*1 array to variable. (FNF)
C   930922  Minor cosmetic change. (FNF)
C***END PROLOGUE  XERRWD
C
C*Internal Notes:
C
C For a different default logical unit number, IXSAV (or a subsidiary
C routine that it calls) will need to be modified.
C For a different run-abort command, change the statement following
C statement 100 at the end.
C-----------------------------------------------------------------------
C Subroutines called by XERRWD.. None
C Function routine called by XERRWD.. IXSAV
C-----------------------------------------------------------------------
C**End
C
C  Declare arguments.
C
      DOUBLE PRECISION R1, R2
      INTEGER NMES, NERR, LEVEL, NI, I1, I2, NR
      CHARACTER*(*) MSG
C
C  Declare local variables.
C
      INTEGER LUNIT, IXSAV, MESFLG
C
C  Get logical unit number and message print flag.
C
C***FIRST EXECUTABLE STATEMENT  XERRWD
      LUNIT = IXSAV (1, 0, .FALSE.)
      MESFLG = IXSAV (2, 0, .FALSE.)
      IF (MESFLG .EQ. 0) GO TO 100
C
```

```
C  Write the message.
C
      WRITE (LUNIT,10)  MSG
 10   FORMAT(1X,A)
      IF (NI .EQ. 1) WRITE (LUNIT, 20) I1
 20   FORMAT(6X,'In above message,  I1 =',I10)
      IF (NI .EQ. 2) WRITE (LUNIT, 30) I1,I2
 30   FORMAT(6X,'In above message,  I1 =',I10,3X,'I2 =',I10)
      IF (NR .EQ. 1) WRITE (LUNIT, 40) R1
 40   FORMAT(6X,'In above message,  R1 =',D21.13)
      IF (NR .EQ. 2) WRITE (LUNIT, 50) R1,R2
 50   FORMAT(6X,'In above,  R1 =',D21.13,3X,'R2 =',D21.13)
C
C  Abort the run if LEVEL = 2.
C
 100  IF (LEVEL .NE. 2) RETURN
      STOP
C----------------------- End of Subroutine XERRWD ----------------------
      END
*DECK XSETF
      SUBROUTINE XSETF (MFLAG)
C***BEGIN PROLOGUE  XSETF
C***PURPOSE  Reset the error print control flag.
C***LIBRARY   MATHLIB
C***CATEGORY  R3A
C***TYPE      ALL (XSETF-A)
C***KEYWORDS  ERROR CONTROL
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C   XSETF sets the error print control flag to MFLAG:
C      MFLAG=1 means print all messages (the default).
C      MFLAG=0 means no printing.
C
C***SEE ALSO  XERMSG, XERRWD, XERRWV
C***REFERENCES  (NONE)
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
C   930329  Added SLATEC format prologue. (FNF)
C   930407  Corrected SEE ALSO section. (FNF)
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  XSETF
C
C Subroutines called by XSETF.. None
C Function routine called by XSETF.. IXSAV
C-----------------------------------------------------------------------
C**End
      INTEGER MFLAG, JUNK, IXSAV
C
C***FIRST EXECUTABLE STATEMENT  XSETF
      IF (MFLAG .EQ. 0 .OR. MFLAG .EQ. 1) JUNK = IXSAV (2,MFLAG,.TRUE.)
      RETURN
C----------------------- End of Subroutine XSETF -----------------------
      END
*DECK XSETUN
      SUBROUTINE XSETUN (LUN)
C***BEGIN PROLOGUE  XSETUN
C***PURPOSE  Reset the logical unit number for error messages.
C***LIBRARY   MATHLIB
C***CATEGORY  R3B
C***TYPE      ALL (XSETUN-A)
C***KEYWORDS  ERROR CONTROL
C***DESCRIPTION
C
C   XSETUN sets the logical unit number for error messages to LUN.
C
```

```
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***SEE ALSO  XERMSG, XERRWD, XERRWV
C***REFERENCES  (NONE)
C***ROUTINES CALLED  IXSAV
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
C   930329  Added SLATEC format prologue. (FNF)
C   930407  Corrected SEE ALSO section. (FNF)
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  XSETUN
C
C Subroutines called by XSETUN.. None
C Function routine called by XSETUN.. IXSAV
C-----------------------------------------------------------------------
C**End
      INTEGER LUN, JUNK, IXSAV
C
C***FIRST EXECUTABLE STATEMENT  XSETUN
      IF (LUN .GT. 0) JUNK = IXSAV (1,LUN,.TRUE.)
      RETURN
C---------------------- End of Subroutine XSETUN ----------------------
      END
*DECK IXSAV
      INTEGER FUNCTION IXSAV (IPAR, IVALUE, ISET)
C***BEGIN PROLOGUE  IXSAV
C***SUBSIDIARY
C***PURPOSE  Save and recall error message control parameters.
C***LIBRARY   MATHLIB
C***CATEGORY  R3C
C***TYPE      ALL (IXSAV-A)
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C
C  IXSAV saves and recalls one of two error message parameters:
C    LUNIT, the logical unit number to which messages are printed, and
C    MESFLG, the message print flag.
C  This is a modification of the SLATEC library routine J4SAVE.
C
C  Saved local variables..
C   LUNIT  = Logical unit number for messages.  The default is obtained
C            by a call to IUMACH (may be machine-dependent).
C   MESFLG = Print control flag..
C            1 means print all messages (the default).
C            0 means no printing.
C
C  On input..
C    IPAR   = Parameter indicator (1 for LUNIT, 2 for MESFLG).
C    IVALUE = The value to be set for the parameter, if ISET = .TRUE.
C    ISET   = Logical flag to indicate whether to read or write.
C             If ISET = .TRUE., the parameter will be given
C             the value IVALUE.  If ISET = .FALSE., the parameter
C             will be unchanged, and IVALUE is a dummy argument.
C
C  On return..
C    IXSAV = The (old) value of the parameter.
C
C***SEE ALSO  XERMSG, XERRWD, XERRWV
C***ROUTINES CALLED  IUMACH
C***REVISION HISTORY  (YYMMDD)
C   921118  DATE WRITTEN
C   930329  Modified prologue to SLATEC format. (FNF)
C   930915  Added IUMACH call to get default output unit.  (ACH)
C   930922  Minor cosmetic changes. (FNF)
C***END PROLOGUE  IXSAV
C
C Subroutines called by IXSAV.. None
C Function routine called by IXSAV.. IUMACH
```

```
C-----------------------------------------------------------------------
C**End
      LOGICAL ISET
      INTEGER IPAR, IVALUE
C-----------------------------------------------------------------------
      INTEGER LUNIT, MESFLG
C-----------------------------------------------------------------------
C The following Fortran-77 declaration is to cause the values of the
C listed (local) variables to be saved between calls to this routine.
C-----------------------------------------------------------------------
      SAVE LUNIT, MESFLG
      DATA LUNIT/-1/, MESFLG/1/
C
C***FIRST EXECUTABLE STATEMENT  IXSAV
      IF (IPAR .EQ. 1) THEN
        IF (LUNIT .EQ. -1) LUNIT = IUMACH()
        IXSAV = LUNIT
        IF (ISET) LUNIT = IVALUE
        ENDIF
C
      IF (IPAR .EQ. 2) THEN
        IXSAV = MESFLG
        IF (ISET) MESFLG = IVALUE
        ENDIF
C
      RETURN
C----------------------- End of Function IXSAV -------------------------
      END
*DECK IUMACH
      INTEGER FUNCTION IUMACH()
C***BEGIN PROLOGUE  IUMACH
C***PURPOSE  Provide standard output unit number.
C***LIBRARY   MATHLIB
C***CATEGORY  R1
C***TYPE      INTEGER (IUMACH-I)
C***KEYWORDS  MACHINE CONSTANTS
C***AUTHOR  Hindmarsh, Alan C., (LLNL)
C***DESCRIPTION
C *Usage:
C        INTEGER  LOUT, IUMACH
C        LOUT = IUMACH()
C
C *Function Return Values:
C     LOUT : the standard logical unit for Fortran output.
C
C***REFERENCES  (NONE)
C***ROUTINES CALLED  (NONE)
C***REVISION HISTORY  (YYMMDD)
C   930915  DATE WRITTEN
C   930922  Made user-callable, and other cosmetic changes. (FNF)
C***END PROLOGUE  IUMACH
C
C*Internal Notes:
C  The built-in value of 6 is standard on a wide range of Fortran
C  systems.  This may be machine-dependent.
C**End
C***FIRST EXECUTABLE STATEMENT  IUMACH
      IUMACH = 6
C
      RETURN
C----------------------- End of Function IUMACH ------------------------
      END

      subroutine dgefa(a,lda,n,ipvt,info)
      integer lda,n,ipvt(1),info
      double precision a(lda,1)
c
```

```
c     dgefa factors a double precision matrix by gaussian elimination.
c
c     dgefa is usually called by dgeco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c     (time for dgeco) = (1 + 9/n)*(time for dgefa) .
c
c     on entry
c
c        a       double precision(lda, n)
c                the matrix to be factored.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       an upper triangular matrix and the multipliers
c                which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that dgesl or dgedi will divide by zero
c                     if called.  use  rcond  in dgeco for a reliable
c                     indication of singularity.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,dscal,idamax
c
c     internal variables
c
      double precision t
      integer idamax,j,k,kp1,l,nm1
c
c
c     gaussian elimination with partial pivoting
c
      info = 0
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 k = 1, nm1
         kp1 = k + 1
c
c        find l = pivot index
c
         l = idamax(n-k+1,a(k,k),1) + k - 1
         ipvt(k) = l
c
c        zero pivot implies this column already triangularized
c
         if (a(l,k) .eq. 0.0d0) go to 40
c
c           interchange if necessary
```

```
c
            if (l .eq. k) go to 10
               t = a(l,k)
               a(l,k) = a(k,k)
               a(k,k) = t
   10       continue
c
c           compute multipliers
c
            t = -1.0d0/a(k,k)
            call dscal(n-k,t,a(k+1,k),1)
c
c           row elimination with column indexing
c
            do 30 j = kp1, n
               t = a(l,j)
               if (l .eq. k) go to 20
                  a(l,j) = a(k,j)
                  a(k,j) = t
   20          continue
               call daxpy(n-k,t,a(k+1,k),1,a(k+1,j),1)
   30       continue
         go to 50
   40    continue
            info = k
   50    continue
   60 continue
   70 continue
      ipvt(n) = n
      if (a(n,n) .eq. 0.0d0) info = n
      return
      end
      subroutine dgesl(a,lda,n,ipvt,b,job)
      integer lda,n,ipvt(1),job
      double precision a(lda,1),b(1)
c
c     dgesl solves the double precision system
c     a * x = b  or  trans(a) * x = b
c     using the factors computed by dgeco or dgefa.
c
c     on entry
c
c        a       double precision(lda, n)
c                the output from dgeco or dgefa.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c        ipvt    integer(n)
c                the pivot vector from dgeco or dgefa.
c
c        b       double precision(n)
c                the right hand side vector.
c
c        job     integer
c                = 0         to solve  a*x = b ,
c                = nonzero   to solve  trans(a)*x = b  where
c                            trans(a)  is the transpose.
c
c     on return
c
c        b       the solution vector  x .
c
c     error condition
```

```
c
c        a division by zero will occur if the input factor contains a
c        zero on the diagonal.  technically this indicates singularity
c        but it is often caused by improper arguments or improper
c        setting of lda .  it will not occur if the subroutines are
c        called correctly and if dgeco has set rcond .gt. 0.0
c        or dgefa has set info .eq. 0 .
c
c     to compute  inverse(a) * c  where  c  is a matrix
c     with  p  columns
c           call dgeco(a,lda,n,ipvt,rcond,z)
c           if (rcond is too small) go to ...
c           do 10 j = 1, p
c              call dgesl(a,lda,n,ipvt,c(1,j),0)
c        10 continue
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,ddot
c
c     internal variables
c
      double precision ddot,t
      integer k,kb,l,nm1
c
      nm1 = n - 1
      if (job .ne. 0) go to 50
c
c        job = 0 , solve  a * x = b
c        first solve  l*y = b
c
         if (nm1 .lt. 1) go to 30
         do 20 k = 1, nm1
            l = ipvt(k)
            t = b(l)
            if (l .eq. k) go to 10
               b(l) = b(k)
               b(k) = t
   10       continue
            call daxpy(n-k,t,a(k+1,k),1,b(k+1),1)
   20    continue
   30    continue
c
c        now solve  u*x = y
c
         do 40 kb = 1, n
            k = n + 1 - kb
            b(k) = b(k)/a(k,k)
            t = -b(k)
            call daxpy(k-1,t,a(1,k),1,b(1),1)
   40    continue
      go to 100
   50 continue
c
c        job = nonzero, solve  trans(a) * x = b
c        first solve  trans(u)*y = b
c
         do 60 k = 1, n
            t = ddot(k-1,a(1,k),1,b(1),1)
            b(k) = (b(k) - t)/a(k,k)
   60    continue
c
c        now solve trans(l)*x = y
c
```

```
         if (nm1 .lt. 1) go to 90
         do 80 kb = 1, nm1
            k = n - kb
            b(k) = b(k) + ddot(n-k,a(k+1,k),1,b(k+1),1)
            l = ipvt(k)
            if (l .eq. k) go to 70
               t = b(l)
               b(l) = b(k)
               b(k) = t
   70       continue
   80    continue
   90    continue
  100 continue
      return
      end
      subroutine dgbfa(abd,lda,n,ml,mu,ipvt,info)
      integer lda,n,ml,mu,ipvt(1),info
      double precision abd(lda,1)
c
c     dgbfa factors a double precision band matrix by elimination.
c
c     dgbfa is usually called by dgbco, but it can be called
c     directly with a saving in time if  rcond  is not needed.
c
c     on entry
c
c        abd     double precision(lda, n)
c                contains the matrix in band storage.  the columns
c                of the matrix are stored in the columns of  abd  and
c                the diagonals of the matrix are stored in rows
c                ml+1 through 2*ml+mu+1 of  abd .
c                see the comments below for details.
c
c        lda     integer
c                the leading dimension of the array  abd .
c                lda must be .ge. 2*ml + mu + 1 .
c
c        n       integer
c                the order of the original matrix.
c
c        ml      integer
c                number of diagonals below the main diagonal.
c                0 .le. ml .lt. n .
c
c        mu      integer
c                number of diagonals above the main diagonal.
c                0 .le. mu .lt. n .
c                more efficient if  ml .le. mu .
c     on return
c
c        abd     an upper triangular matrix in band storage and
c                the multipliers which were used to obtain it.
c                the factorization can be written  a = l*u  where
c                l  is a product of permutation and unit lower
c                triangular matrices and  u  is upper triangular.
c
c        ipvt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if  u(k,k) .eq. 0.0 .  this is not an error
c                     condition for this subroutine, but it does
c                     indicate that dgbsl will divide by zero if
c                     called.  use  rcond  in dgbco for a reliable
c                     indication of singularity.
c
```

```
c     band storage
c
c           if  a  is a band matrix, the following program segment
c           will set up the input.
c
c                   ml = (band width below the diagonal)
c                   mu = (band width above the diagonal)
c                   m = ml + mu + 1
c                   do 20 j = 1, n
c                      i1 = max0(1, j-mu)
c                      i2 = min0(n, j+ml)
c                      do 10 i = i1, i2
c                         k = i - j + m
c                         abd(k,j) = a(i,j)
c                10    continue
c                20 continue
c
c           this uses rows  ml+1  through  2*ml+mu+1  of  abd .
c           in addition, the first  ml  rows in  abd  are used for
c           elements generated during the triangularization.
c           the total number of rows needed in  abd  is  2*ml+mu+1 .
c           the  ml+mu by ml+mu  upper left triangle and the
c           ml by ml  lower right triangle are not referenced.
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,dscal,idamax
c     fortran max0,min0
c
c     internal variables
c
      double precision t
      integer i,idamax,i0,j,ju,jz,j0,j1,k,kp1,l,lm,m,mm,nm1
c
c
      m = ml + mu + 1
      info = 0
c
c     zero initial fill-in columns
c
      j0 = mu + 2
      j1 = min0(n,m) - 1
      if (j1 .lt. j0) go to 30
      do 20 jz = j0, j1
         i0 = m + 1 - jz
         do 10 i = i0, ml
            abd(i,jz) = 0.0d0
   10    continue
   20 continue
   30 continue
      jz = j1
      ju = 0
c
c     gaussian elimination with partial pivoting
c
      nm1 = n - 1
      if (nm1 .lt. 1) go to 130
      do 120 k = 1, nm1
         kp1 = k + 1
c
c        zero next fill-in column
c
         jz = jz + 1
         if (jz .gt. n) go to 50
```

```
         if (ml .lt. 1) go to 50
            do 40 i = 1, ml
               abd(i,jz) = 0.0d0
   40       continue
   50    continue
c
c        find l = pivot index
c
         lm = min0(ml,n-k)
         l = idamax(lm+1,abd(m,k),1) + m - 1
         ipvt(k) = l + k - m
c
c        zero pivot implies this column already triangularized
c
         if (abd(l,k) .eq. 0.0d0) go to 100
c
c           interchange if necessary
c
            if (l .eq. m) go to 60
               t = abd(l,k)
               abd(l,k) = abd(m,k)
               abd(m,k) = t
   60       continue
c
c           compute multipliers
c
            t = -1.0d0/abd(m,k)
            call dscal(lm,t,abd(m+1,k),1)
c
c           row elimination with column indexing
c
            ju = min0(max0(ju,mu+ipvt(k)),n)
            mm = m
            if (ju .lt. kp1) go to 90
            do 80 j = kp1, ju
               l = l - 1
               mm = mm - 1
               t = abd(l,j)
               if (l .eq. mm) go to 70
                  abd(l,j) = abd(mm,j)
                  abd(mm,j) = t
   70          continue
               call daxpy(lm,t,abd(m+1,k),1,abd(mm+1,j),1)
   80       continue
   90       continue
         go to 110
  100    continue
            info = k
  110    continue
  120 continue
  130 continue
      ipvt(n) = n
      if (abd(m,n) .eq. 0.0d0) info = n
      return
      end
      subroutine dgbsl(abd,lda,n,ml,mu,ipvt,b,job)
      integer lda,n,ml,mu,ipvt(1),job
      double precision abd(lda,1),b(1)
c
c     dgbsl solves the double precision band system
c     a * x = b  or  trans(a) * x = b
c     using the factors computed by dgbco or dgbfa.
c
c     on entry
c
c        abd     double precision(lda, n)
c                the output from dgbco or dgbfa.
```

```
c
c        lda     integer
c                the leading dimension of the array  abd .
c
c        n       integer
c                the order of the original matrix.
c
c        ml      integer
c                number of diagonals below the main diagonal.
c
c        mu      integer
c                number of diagonals above the main diagonal.
c
c        ipvt    integer(n)
c                the pivot vector from dgbco or dgbfa.
c
c        b       double precision(n)
c                the right hand side vector.
c
c        job     integer
c                = 0         to solve  a*x = b ,
c                = nonzero   to solve  trans(a)*x = b , where
c                            trans(a)  is the transpose.
c
c     on return
c
c        b       the solution vector  x .
c
c     error condition
c
c        a division by zero will occur if the input factor contains a
c        zero on the diagonal.  technically this indicates singularity
c        but it is often caused by improper arguments or improper
c        setting of lda .  it will not occur if the subroutines are
c        called correctly and if dgbco has set rcond .gt. 0.0
c        or dgbfa has set info .eq. 0 .
c
c     to compute  inverse(a) * c  where  c  is a matrix
c     with  p  columns
c           call dgbco(abd,lda,n,ml,mu,ipvt,rcond,z)
c           if (rcond is too small) go to ...
c           do 10 j = 1, p
c              call dgbsl(abd,lda,n,ml,mu,ipvt,c(1,j),0)
c        10 continue
c
c     linpack. this version dated 08/14/78 .
c     cleve moler, university of new mexico, argonne national lab.
c
c     subroutines and functions
c
c     blas daxpy,ddot
c     fortran min0
c
c     internal variables
c
      double precision ddot,t
      integer k,kb,l,la,lb,lm,m,nm1
c
      m = mu + ml + 1
      nm1 = n - 1
      if (job .ne. 0) go to 50
c
c        job = 0 , solve  a * x = b
c        first solve l*y = b
c
         if (ml .eq. 0) go to 30
         if (nm1 .lt. 1) go to 30
```

```
            do 20 k = 1, nm1
               lm = min0(ml,n-k)
               l = ipvt(k)
               t = b(l)
               if (l .eq. k) go to 10
                  b(l) = b(k)
                  b(k) = t
   10          continue
               call daxpy(lm,t,abd(m+1,k),1,b(k+1),1)
   20       continue
   30    continue
c
c        now solve  u*x = y
c
         do 40 kb = 1, n
            k = n + 1 - kb
            b(k) = b(k)/abd(m,k)
            lm = min0(k,m) - 1
            la = m - lm
            lb = k - lm
            t = -b(k)
            call daxpy(lm,t,abd(la,k),1,b(lb),1)
   40    continue
      go to 100
   50 continue
c
c        job = nonzero, solve  trans(a) * x = b
c        first solve  trans(u)*y = b
c
         do 60 k = 1, n
            lm = min0(k,m) - 1
            la = m - lm
            lb = k - lm
            t = ddot(lm,abd(la,k),1,b(lb),1)
            b(k) = (b(k) - t)/abd(m,k)
   60    continue
c
c        now solve trans(l)*x = y
c
         if (ml .eq. 0) go to 90
         if (nm1 .lt. 1) go to 90
            do 80 kb = 1, nm1
               k = n - kb
               lm = min0(ml,n-k)
               b(k) = b(k) + ddot(lm,abd(m+1,k),1,b(k+1),1)
               l = ipvt(k)
               if (l .eq. k) go to 70
                  t = b(l)
                  b(l) = b(k)
                  b(k) = t
   70          continue
   80       continue
   90    continue
  100 continue
      return
      end
      subroutine daxpy(n,da,dx,incx,dy,incy)
c
c     constant times a vector plus a vector.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dy(1),da
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if (da .eq. 0.0d0) return
```

```
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dy(iy) = dy(iy) + da*dx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,4)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dy(i) = dy(i) + da*dx(i)
   30 continue
      if( n .lt. 4 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,4
        dy(i) = dy(i) + da*dx(i)
        dy(i + 1) = dy(i + 1) + da*dx(i + 1)
        dy(i + 2) = dy(i + 2) + da*dx(i + 2)
        dy(i + 3) = dy(i + 3) + da*dx(i + 3)
   50 continue
      return
      end
      subroutine dscal(n,da,dx,incx)
c
c     scales a vector by a constant.
c     uses unrolled loops for increment equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision da,dx(1)
      integer i,incx,m,mp1,n,nincx
c
      if(n.le.0)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      nincx = n*incx
      do 10 i = 1,nincx,incx
        dx(i) = da*dx(i)
   10 continue
      return
c
c        code for increment equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dx(i) = da*dx(i)
   30 continue
      if( n .lt. 5 ) return
```

```
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dx(i) = da*dx(i)
        dx(i + 1) = da*dx(i + 1)
        dx(i + 2) = da*dx(i + 2)
        dx(i + 3) = da*dx(i + 3)
        dx(i + 4) = da*dx(i + 4)
   50 continue
      return
      end
      double precision function ddot(n,dx,incx,dy,incy)
c
c     forms the dot product of two vectors.
c     uses unrolled loops for increments equal to one.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dy(1),dtemp
      integer i,incx,incy,ix,iy,m,mp1,n
c
      ddot = 0.0d0
      dtemp = 0.0d0
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        dtemp = dtemp + dx(ix)*dy(iy)
        ix = ix + incx
        iy = iy + incy
   10 continue
      ddot = dtemp
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,5)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        dtemp = dtemp + dx(i)*dy(i)
   30 continue
      if( n .lt. 5 ) go to 60
   40 mp1 = m + 1
      do 50 i = mp1,n,5
        dtemp = dtemp + dx(i)*dy(i) + dx(i + 1)*dy(i + 1) +
     *   dx(i + 2)*dy(i + 2) + dx(i + 3)*dy(i + 3) + dx(i + 4)*dy(i + 4)
   50 continue
   60 ddot = dtemp
      return
      end
      integer function idamax(n,dx,incx)
c
c     finds the index of element having max. absolute value.
c     jack dongarra, linpack, 3/11/78.
c
      double precision dx(1),dmax
      integer i,incx,ix,n
c
      idamax = 0
```

```
      if( n .lt. 1 ) return
      idamax = 1
      if(n.eq.1)return
      if(incx.eq.1)go to 20
c
c        code for increment not equal to 1
c
      ix = 1
      dmax = dabs(dx(1))
      ix = ix + incx
      do 10 i = 2,n
         if(dabs(dx(ix)).le.dmax) go to 5
         idamax = i
         dmax = dabs(dx(ix))
    5    ix = ix + incx
   10 continue
      return
c
c        code for increment equal to 1
c
   20 dmax = dabs(dx(1))
      do 30 i = 2,n
         if(dabs(dx(i)).le.dmax) go to 30
         idamax = i
         dmax = dabs(dx(i))
   30 continue
      return
      end
      subroutine dcopy(n,sx,incx,sy,incy)
c
c     copies a vector, x, to a vector, y.
c     uses unrolled loops for increments equal to 1.
c     jack dongarra, linpack, 3/11/78.
c
      double precision sx(1),sy(1)
      integer i,incx,incy,ix,iy,m,mp1,n
c
      if(n.le.0)return
      if(incx.eq.1.and.incy.eq.1)go to 20
c
c        code for unequal increments or equal increments
c          not equal to 1
c
      ix = 1
      iy = 1
      if(incx.lt.0)ix = (-n+1)*incx + 1
      if(incy.lt.0)iy = (-n+1)*incy + 1
      do 10 i = 1,n
        sy(iy) = sx(ix)
        ix = ix + incx
        iy = iy + incy
   10 continue
      return
c
c        code for both increments equal to 1
c
c
c        clean-up loop
c
   20 m = mod(n,7)
      if( m .eq. 0 ) go to 40
      do 30 i = 1,m
        sy(i) = sx(i)
   30 continue
      if( n .lt. 7 ) return
   40 mp1 = m + 1
      do 50 i = mp1,n,7
```

```
      sy(i) = sx(i)
      sy(i + 1) = sx(i + 1)
      sy(i + 2) = sx(i + 2)
      sy(i + 3) = sx(i + 3)
      sy(i + 4) = sx(i + 4)
      sy(i + 5) = sx(i + 5)
      sy(i + 6) = sx(i + 6)
50 continue
   return
   end
```