REVISTA MATEMÁTICA IBEROAMERICANA
VOL. 2, N.^{os} 1 Y 2, 1986

Relativistic Stability of Matter-I

C. Fefferman* and R. de la Llave**

Dedicated to A. P. Calderón

0. Introduction

In this article, we study the quantum mechanics of N electrons and M nuclei interacting by Coulomb forces. Motivated by an important idea of Chandrasekhar and following Herbst [H], we modify the usual kinetic energy $-\Delta$ to take into account an effect from special relativity. As a result, the system can implode for unfavorable values of the nuclear charge Z and the fine structure constant α. This is analogus to the gravitational collapse of a heavy star. Our goal here is to find those values of α and Z for which the system is stable.

We start by explaining the modified kinetic energy. Recall that $-\Delta$ is the quantum version of the kinetic energy $T = p^2/2m$ from classical mechanics. Now $T \sim p^2/2m$ is only a low-energy approximation in special relativity. The correct formula is $T = (p^2 + m^2)^{1/2} - m$ in suitable units. In particular, $T \sim |p|$ in the crucial high-energy limit. A first crude attempt to reflect this in quantum mechanics is to study the Hamiltonian

$$H = [(-\Delta + m^2)^{1/2} - m] + V(x) \qquad \text{(A)}$$

in place of the usual $-\Delta + V(x)$. While (A) is not a correct physical equation, it is rooted in a genuine relativistic effect. It is essentially equivalent to the ideas that led Chandrasekhar to predict the gravitational collapse of stars and

* Research supported in part by NSF Grants MCS80-03072 and DMS85-04342.
** Research supported in part by NSF Grants MCS82-01604 and DmS85-04984.

derive the order of magnitude of the critical mass. (See also Lieb-Thirring [LT2].)

Next we show how a system governed by (A) can implode. This happens already for a single quantized electron attracted to a single nucleus of charge Z fixed at the origin. The Hamiltonian (A) becomes $[(-\Delta + m^2)^{1/2} - m] - \alpha Z/|x|$, where $\alpha \sim 1/137$ is the fine structure constant. Modulo a bounded error,

$$H = (-\Delta)^{1/2} - \frac{\alpha Z}{|x|} \quad \text{acting on} \quad L^2(\mathbb{R}^3) \tag{B}$$

The state of the electron is given by a wave function $\psi \in L^2(\mathbb{R}^3)$ of norm 1, and the energy is $\langle H\psi, \psi\rangle$. Unlike the usual Schrödinger operator, (B) is homogeneous with respect to dilations of $\mathbb{R}^3$. So if we could find a $\varphi \in C_0^\infty$ with $\langle H\varphi, \varphi\rangle < 0$, then by dilating φ we can make wave functions ψ concentrated in a tiny region of space, and having large negative energy. In other words, the electron will fall into the nucleus. On the other hand, if $\langle H\varphi, \varphi\rangle > 0$, then dilating φ to concentrate the electron into a small region will make the energy large positive. Consequently, the system will be stable if $H \geqslant 0$. From these considerations, we may expect the value of αZ to play a crucial role in the study of (B). In fact, (B) is easily decomposed by means of spherical harmonics and the Mellin transform, and one finds the following dichotomy: $H \geqslant 0$ if $\alpha Z \leqslant 2/\pi$; H unbounded below if $\alpha Z > 2/\pi$. See Herbst [H]. Putting $\alpha = 1/137$, we obtain $2/\pi\alpha \sim 87$ as the largest stable nuclear charge. Of course larger atoms exist in nature; numbers predicted from (A), (B) are physically relevant only as orders of magnitude.

We want to understand what happens for N electrons and M nuclei. The set-up is as follows. We fix nuclei with charges $Z_1, Z_2, \ldots, Z_M$ at points $y_1 \ldots y_M \in \mathbb{R}^3$. A quantum state for the N electrons amounts to a wave function $\psi(x_1 \ldots x_N) \in L^2(\mathbb{R}^{3N})$ of norm 1. Since electrons are Fermions, we take ψ antisymmetric, i.e., $\psi(x_1 \ldots x_N) = \operatorname{sgn}(\pi)\psi(x_{\pi 1} \ldots x_{\pi N})$ for permutations π. Note that our wave functions are scalars, i.e., we are ignoring spin. The energy of a quantum state ψ is defined as $\langle H\psi, \psi\rangle$ with

$$H = \sum_{k=1}^{N} (-\Delta_{x_k})^{1/2} + \alpha\left[\sum_{j<k} \frac{1}{|x_j - x_k|} + \sum_{j<k} \frac{Z_j Z_k}{|y_j - y_k|} - \sum_{j,k} \frac{Z_k}{|x_j - y_k|}\right]. \tag{C}$$

This is the many-particle version of (B), and agrees with (A) up to an error $O(N)$. To make sure $H\psi$ is well-defined, we take $\psi \in C_0^\infty$.

Depending on α, $Z_1 \ldots Z_M$, three very different phenomena may occur.

Case 1 (Stability): $\langle H\psi, \psi\rangle \geqslant 0$ for all ψ and all $y_1 \ldots y_M$.

Case 2 (Many-body implosion): $\langle H\psi, \psi\rangle$ is bounded below for each fixed $y_1 \ldots y_M$, but the ground-state energy tends to $-\infty$ as the nuclei are bought together at the origin in a suitable manner.

Case 3 (Complete implosion): $\langle H\psi, \psi\rangle$ is unbounded below for each fixed $y_1 \ldots y_M$.

That these are the only possible cases is immediate from dilation-invariance of the Hamiltonian (C).

From the study of (B) one checks easily that complete implosion occurs if and only if $\alpha \cdot \max_k Z_k > 2/\pi$. The other two cases are hard to resolve.

Stability (case 1) is a strong statement about many-body systems. For instance, by applying the trivial estimate $(-\Delta)^{1/2} \leqslant \alpha(-\Delta) + \alpha^{-1}/4$, we get in case 1

$$\sum_k (-\Delta_{x_k}) + \sum_{j<k} \frac{1}{|x_j - x_k|} + \sum_{j<k} \frac{Z_j Z_k}{|y_j - y_k|} - \sum_{j,k} \frac{Z_k}{|x_j - y_k|} \geqslant -CN \qquad \text{(D)}$$

with $C = \alpha^{-2}/4$. This is a sharp form of stability of matter (see Dyson-Lenard [DL1], Federbush [F], Lieb-Thirring [LT]). Antisymmetry of the wave function must therefore play a crucial role in stability of (C). Already in the case of a single electron ($N = 1$), (D) is not obvious; its analogue in Thomas-Fermi theory [L] is false.

In fact, the case of one electron and M nuclei was settled by Daubechies and Lieb [DL2] for $\alpha \leqslant 1/3\pi$. They found that the Hamiltonian (C) is stable in this case under the sharp assumption $\alpha \cdot \max_k Z_k \leqslant 2/\pi$. It would be interesting to know the best constant in (*D*) with $N = 1$.

Recently, J. Conlon [C] proved the further deep result that the Hamiltonian (C) is stable for arbitrarily many electrons and nuclei, provided $\alpha \cdot \max_k Z_k \leqslant 10^{-200}$.

Both [DL2] and [C] are major steps forward in our understanding of relativistic stability of matter. They contain a wealth of important insights which we take for granted here.

Our results are as follows.

Theorem 1 (Hydrogen). *Suppose all* $Z_k = 1$. *Then the Hamiltonian* (C) *is stable for* $\alpha \leqslant 1/2.06\pi$.

Theorem 2 (Small α). *There exists* α_{critical} *such that* $\alpha < \alpha_{\text{critical}}$ *and* $\alpha \max_k Z_k \leqslant 2/\pi$ *imply stability of* (C).

Since complete implosion occurs when $\alpha \cdot \max_k Z_k > 2/\pi$, Theorem 1 provides the best α up to a factor of 4.12, while Theorem 2 completely settles the

case of small α. From the proof of Theorem 2 one has a good chance to find an explicit (non-optimal) α_{critical}. Moreover, we hope that $\alpha_{\text{critical}} > 1/137$. H. Trotter has produced strong evidence for this, but a complete proof is not yet available.

We should point out here two immediate consequences of our proof, which may be of independent interest.

Cor. 1. *Let $x_1 \dots x_N, y_1 \dots y_M \in \mathbb{R}^3$ be electrons and protons, let $\delta(x_k)$ be the distance from x_k to the nearest y_l, and let*

$$V = \sum_{j<k} |x_j - x_k|^{-1} + \sum_{j<k} |y_j - y_k|^{-1} - \sum_{j,k} |x_j - y_k|^{-1}$$

be the Coulomb potential. Then

$$V \geqslant -\frac{8}{3} \sum_k \frac{1}{\delta(x_k)}.$$

Note. Elliott Lieb has pointed out to us that Cor. 1. appears in Baxter [B], 1980.

Cor. 2. **Let** *$\psi(x_1 \dots x_N)$ be antisymmetric, and let $\delta(x_k)$ be the distance from x_k to the nearest $x_l (l \neq k)$.* Then

$$\left\langle \sum_k (-\Delta_{x_k})^{1/2} \psi, \psi \right\rangle \geqslant \frac{35}{24\pi} \left\langle \sum_k \frac{1}{\delta(x_k)} \psi, \psi \right\rangle.$$

More generally, we hope that the ideas in our proofs will be useful tools for understanding many-body problems.

In this paper, we prove only Theorem 1, leaving Theorem 2 for later in the hope of getting $\alpha_{\text{critical}} > 1/137$. An intresting feature of our proof of Theorem 1 is that it makes use of computers. This is perhaps natural in a theorem on best constants. (Without computers, we can only prove Theorem 1 with $1/2.06\pi$ replaced by $1/5\pi$.)

We begin our proof by reducing Theorem 1 to a problem involving only one electron and three nuclei. The reduction of a «big» problem for N electrons and M nuclei to a «small» problem for a few particles is made possible by the key equations (1), (2), (6) below. The one-electron problem in turn can be reduced to the non-vanishing of a function $D(t_1, t_2)$ defined in terms of the solution of a second-order linear ordinary differential equation. Since the equation can be solved numerically, it is easy to calculate $D(t_1, t_2)$ to good accuracy with a computer and convince oneself that $D(t_1, t_2) \neq 0$. However, a complete proof requires computer programs that produce rigorous upper and lower bounds for solutions of ordinary differential equations. Related computer-science issues arose in the rigorous study of the renormalization group equations (see [La, Ll]).

In section 6 below, we explain the compuer programs in detail, and provide listings so that our results may be easily reproduced.

1. Notation

$\psi(x_1 \dots x_s)$ denotes a many-electron wave function, while $y_1 \dots y_s$, are fixed nuclei in $\mathbb{R}^3$.

$B(z, R)$ = ball of center z and radius R in $\mathbb{R}^3$.

$N(z, R) = N(z, R; x_1 \dots x_s)$ = number of electrons x_j in $B(z, R)$.

$M(z, R)$ = number of nuclei in $B(z, R)$.

Let A be a subset of $\{1 \dots s\}$. We write $x_A = (x_j)_{j \in A}$ and $x'_A = (x_j)_{j \notin A}$. Thus $\psi(x_1 \dots x_s) = \psi(x_A, x'_A)$.

We write dx_A for $\prod_{j \in A} dx_j$, and dx'_A for $\prod_{j \notin A} dx_j$.

If $A = \{k\}$ has just one element, then we write x_k, x'_k, dx'_k, in place of x_A, x'_A, dx'_A. Thus $\psi(x_1 \dots x_s) = \psi(x_k, x'_k)$ for each k.

For a constant $\varkappa$ to be picked later, set

$$H = \frac{12\varkappa}{35} \cdot \frac{\pi}{2} \sum_k (-\Delta_{x_k})^{1/2} + V$$

$$V = \sum_{j<k} \frac{1}{|x_j - x_k|} + \sum_{j<k} \frac{1}{|y_j - y_k|} - \sum_{j,k} \frac{1}{|x_j - y_k|}.$$

Our present H is proportional to the Hamiltonian (C) in the introduction, with $\alpha = (2/\pi) \cdot (35/12\varkappa)$.

2. Rewriting the Hamiltonian

A key idea in our proof is to associate both a kinetic and a potential energy to each ball $B(z,R)$, and then regard the total energy $\langle H\psi, \psi \rangle$ as an integral over all possible balls. To do this for kinetic energy, we use the elementary identities

$$\int_{z \in \mathbb{R}^3} \int_{R>0} \int_{x,y \in B(z,R)} |u(x) - u(y)|^2 \, dx \, dy \frac{dz \, dR}{R^8}$$

$$= \frac{16\pi}{35} \int \frac{|u(x) - u(y)|^2}{|x - y|^4} dx \, dy = \frac{32\pi^3}{35} \langle (-\Delta)^{1/2} u, u \rangle, \; u \in C_0^\infty(\mathbb{R}^3).$$

(The second equality is well-known from the Fourier transform.)

Hence if we set

$$T_k(z, R; x'_k) = \int_{\bar{x}, \bar{\bar{x}} \in B(z,R)} |\psi(\bar{x}, x'_k) - \psi(\bar{\bar{x}}, x'_k)|^2 \, d\bar{x} \, d\bar{\bar{x}}$$

for $\psi \in L^2(\mathbb{R}^{3s})$ and $x'_k \in \mathbb{R}^{3s-3}$, then we have

$$\frac{12\varkappa}{35}\frac{\pi}{2}\sum_{k=1}^{s}\langle(-\Delta_{x_k})^{1/2}\psi,\psi\rangle$$

$$=\frac{3\varkappa}{16\pi^2}\sum_{k=1}^{s}\int_{z\in\mathbb{R}^3}\int_{R>0}\int_{x'_k\in\mathbb{R}^{3s-3}}T_k(z,R;x'_k)\,dx'_k\frac{dz\,dR}{R^8}. \tag{1}$$

To write the potential energy as an integral over spheres, we just note that

$$\frac{1}{|x-x'|}=\frac{1}{\pi}\int_{z\in\mathbb{R}^3}\int_{R>0}\chi_{x,x'\in B(z,R)}\frac{dz\,dR}{R^5}\quad\text{for}\quad x,x'\in\mathbb{R}^3.$$

Summing this over all particle pairs with an appropriate sign, we have at once

$$V=\sum_{j<k}\frac{1}{|x_j-x_k|}+\sum_{j<k}\frac{1}{|y_j-y_k|}-\sum_{j,k}\frac{1}{|x_j-y_k|}$$

$$=\frac{1}{\pi}\int_{x\in\mathbb{R}^3}\int_{R<0}\left\{\frac{N(N-1)}{2}+\frac{M(M-1)}{2}-MN\right\}\frac{dz\,dR}{R^5} \tag{2}$$

where $N=N(z,R;x_1\ldots x_s)$ and $M=M(z,R)$.

Next we bring in antisymmetry of the wave function to prove the following estimate, which we shall put into (1).

Lemma 1. *Fix $B=B(z,R)$, and suppose $\psi(x_1\ldots x_s)$ is antisymmetric. Then*

$$\sum_k\int_{x'_k\in\mathbb{R}^{3s-3}}T_k(z,R;x'_k)\,dx'_k\geqslant\sum_k\int_{x'_k\in(\mathbb{R}^3\setminus B)^{s-1}}T_k(z,R;x'_k)\,dx'_k$$

$$+\frac{8\pi}{3}R^3\int_{\mathbb{R}^{3s}}(N(z,R;x_1\ldots x_s)-1)_+\,|\psi(x_1\ldots x_s)|^2\,dx_1\ldots dx_s.$$

PROOF. Let $\varphi_0,\varphi_1,\varphi_2,\ldots$ be a complete orthonormal system in $L^2(B)$, with $\varphi_0=\text{const}$. If we expand a given $u\in L^2(B)$ into its Fourier series $u(x)\sim$ $\sim\sum A_\alpha\varphi_\alpha(x)$, then

$$\int_{\bar{x},\bar{\bar{x}}\in B}|u(\bar{x})-u(\bar{\bar{x}})|^2\,d\bar{x}\,d\bar{\bar{x}}=\frac{8\pi}{3}R^3\int_{x\in B}|u(x)-Av_Bu|^2\,dx$$

$$=\frac{8\pi}{3}R^3\sum_{\alpha\neq 0}|A_\alpha|^2.$$

Now let $\varphi(x_1\ldots x_N)\in L^2(B^N)$, and write

$$\varphi(x_1\ldots x_N)\sim\sum_{\alpha_1\ldots\alpha_N}A_{\alpha_1\ldots\alpha_N}\varphi_{\alpha_1}(x_1)\ldots\varphi_{\alpha_N}(x_N).$$

The preceding identity gives at once

$$\int_{(x_1,\ldots,x_{k-1},x_{k+1},\ldots,x_N)\in B^{N-1}} \int_{\bar{x},\bar{\bar{x}}\in B} |\varphi(x_1\ldots x_{k-1}\bar{x}x_{k+1}\ldots x_N) - \varphi(x_1\ldots x_{k-1}\bar{\bar{x}}x_{k+1}\ldots x_N)|^2\, d\bar{x}\, d\bar{\bar{x}}\, dx_1\ldots dx_{k-1}\, dx_{k+1}\ldots dx_N$$

$$= \frac{8\pi}{3} R^3 \sum_{\substack{\alpha_1\ldots\alpha_N \\ (\alpha_k\neq 0)}} |A_{\alpha_1\ldots\alpha_N}|^2$$

for each k. Sum this over k, and we get

$$\sum_k \int_{B^{N-1}} \int_{\bar{x},\bar{\bar{x}}\in B} |\varphi(x_1\ldots x_{k-1}\bar{x}x_{k+1}\ldots x_N) - \varphi(x_1\ldots x_{k-1}\bar{\bar{x}}x_{k+1}\ldots x_N)|^2 \, d\bar{x}\, d\bar{\bar{x}}\, dx_1\ldots dx_{k-1}\, dx_{k+1}\ldots dx_N$$

$$= \frac{8\pi}{3} R^3 \sum_{\alpha_1\ldots\alpha_N} |A_{\alpha_1\ldots\alpha_N}|^2 \mathfrak{N}(\alpha_1\ldots\alpha_N), \quad (3)$$

where $\mathfrak{N}(\alpha_1\ldots\alpha_N)$ = (number of k with $\alpha_k\neq 0$).

If $\varphi(x_1\ldots x_N)$ is antisymmetric, then the $A_{\alpha_1\ldots\alpha_N}$ are antisymmetric, so that $\mathfrak{N}(\alpha_1\ldots\alpha_N)\geqslant N-1$ whenever $A_{\alpha_1\ldots\alpha_N}\neq 0$. Hence (3) yields

$$\sum_k \int_{B^{N-1}} \int_{\bar{x},\bar{\bar{x}}\in B} |\varphi(x_1\ldots x_{k-1}\bar{x}x_{k+1}\ldots x_N) - \varphi(x_1\ldots x_{k-1}\bar{\bar{x}}x_{k+1}\ldots x_N)|^2 \, d\bar{x}\, d\bar{\bar{x}}\, dx_1\ldots dx_{k-1}\, dx_{k+1}\ldots dx_N$$

$$\geqslant \frac{8\pi}{3} R^3(N-1) \sum_{\alpha_1\ldots\alpha_N} |A_{\alpha_1\ldots\alpha_N}|^2$$

$$= \frac{8\pi}{3} R^3(N-1) \int_{B^N} |\varphi(x_1\ldots x_N)|^2\, dx_1\ldots dx_N, \quad (4)$$

whenever $\varphi\in L^2(B^N)$ is antisymmetric.

Next, let $\psi(x_1\ldots x_s)\in L^2(\mathbb{R}^{3s})$ be antisymmetric, and let $N\geqslant 2$. For each N-element subset $A\subset\{1\ldots s\}$, let $E(A)=\{(x_1\ldots x_s)\mid x_j\in B$ if and only if $j\in A\}$, and $F(A,k)=\{x'_k=(x_j)_{j\neq k}\mid$ Each $x_j(j\neq k)$ belongs to B if and only if $j\in A\}$. For fixed A and fixed x'_A, we regard

$$\psi|_{E(A)} = \psi(x_A, x'_A)\Big|_{\substack{x_A\in B^N \\ x'_A\in[{}^cB]^{s-N}}}$$

as a function of $x_A\in B^N$, and apply (4). Integrating against dx'_A over $[{}^cB]^{s-N}$, we obtain

$$\sum_{k\in A} \int_{x'_k\in F(A,k)} \int_{\bar{x},\bar{\bar{x}}\in B} |\psi(\bar{x},x'_k) - \psi(\bar{\bar{x}},x'_k)|^2\, d\bar{x}\, d\bar{\bar{x}}\, dx'_k$$

$$\geqslant \frac{8\pi}{3} R^3(N-1) \int_{E(A)} |\psi(x_1\ldots x_s)|^2\, dx_1\ldots dx_s.$$

Sum this over all N-element sets A, and we find that

$$\sum_k \int_{x'_k \in G(N,k)} \int_{\bar{x},\bar{\bar{x}} \in B} |\psi(\bar{x}, x'_k) - \psi(\bar{\bar{x}}, x'_k)|^2 \, d\bar{x}\, d\bar{\bar{x}}\, dx'_k$$

$$\geqslant \frac{8\pi}{3} R^3 (N-1) \int_{\mathbb{R}^{3s}} \chi_{N(z,R;x_1 \ldots x_s)=N} |\psi(x_1 \ldots x_s)|^2 \, dx_1 \ldots dx_s \quad (5)$$

with $G(N,k) = \{x'_k \in \mathbb{R}^{3s-3} \mid$ Exactly $N-1$ of the components of x'_k belong to $B\}$.

If finally, we sum (5) over all N from 2 to s, then we get the conclusion of the Lemma. □

Corollary. *If $\psi(x_1 \ldots x_s)$ is antisymmetric, then*

$$\frac{12\varkappa}{35} \frac{\pi}{2} \sum_{k=1}^{s} \langle (-\Delta_{x_k})^{1/2} \psi, \psi \rangle$$

$$\geqslant \left\langle \frac{1}{2\pi} \int_{z \in \mathbb{R}^3} \int_{R>0} \varkappa [N(z,R;\bullet) - 1]_+ \frac{dz\, dR}{R^5} \psi, \psi \right\rangle$$

$$+ \sum_k \frac{3\varkappa}{16\pi^2} \int_{z \in \mathbb{R}^3} \int_{B>0} \int_{x'_k \in [{}^cB(z,R)]^{s-1}} T_k(z,R;x'_k)\, dx'_k \frac{dz\, dR}{R^8} \quad (6)$$

PROOF. Immediate from the Lemma and (1). □

We pause to point out some simple applications of (2) and (6). (These were stated in the introduction as Corollaries 1 and 2.) Since

$$\int_{z \in \mathbb{R}^3} \int_{R>0} [N(z,R) - 1]_+ \frac{dz\, dR}{2}$$

$$\geqslant \int_{z \in \mathbb{R}^3} \int_{R>0} \chi_{N(z,R) \geq 2} \frac{N(z,R)}{2} \frac{dz\, dR}{R^5}$$

$$= \frac{1}{2} \sum_k \int_{z \in \mathbb{R}^3} \int_{R>0} \chi_{N(z,R) \geq 2} \chi_{x_k \in B(z,R)} \frac{dz\, dR}{R^5}$$

$$\geqslant \frac{1}{2} \sum_k \int_{z \in \mathbb{R}^3} \int_{R>0} \chi_{x_k,\, x_{j(k)} \in B(z,R)} \frac{dz\, dR}{R^5}$$

($x_{j(k)}$ = nearest neighbor of x_k)

$$= \frac{\pi}{2} \sum_k |x_k - x_{j(k)}|^{-1}$$

by (2), it follows from (6) that

$$\frac{12\varkappa}{35}\cdot\frac{\pi}{2}\sum_{k=1}^{s}\langle(-\Delta_{x_k})^{1/2}\psi,\psi\rangle\geqslant\left\langle\frac{\varkappa}{2\pi}\cdot\frac{\pi}{2}\sum_k|x_k-x_{j(k)}|^{-1}\psi,\psi\right\rangle.$$

Setting $\varkappa=4$ and $\delta(x_k)=\min_{j\neq k}|x_j-x_k|$, we obtain

$$\frac{24\pi}{35}\sum_{k=1}^{s}\langle(-\Delta_{x_k})^{1/2}\psi,\psi\rangle\geqslant\left\langle\sum_k\frac{1}{\delta(x_k)}\psi,\psi\right\rangle$$

for antisymmetric ψ. This gives a new quantitative meaning to the intuition that kinetic energy keeps Fermions apart. Similarly, if $\delta(y_k)$ denotes the distance from y_k to the nearest electron then for particles in fixed positions we deduce from (2) that

$$\begin{aligned}
V&=\frac{1}{2\pi}\int_{z\in\mathbb{R}^3}\int_{R>0}\{N(N-1)+M(M-1)-2MN\}\frac{dz\,dR}{R^5}\\
&=\frac{1}{2\pi}\int_{z\in\mathbb{R}^3}\int_{R>0}\left\{\left[\left(N-M-\frac{1}{2}\right)^2-\frac{1}{4}\right]-2M\right\}\frac{dz\,dR}{R^5}\\
&\geqslant\frac{1}{2\pi}\int_{z\in\mathbb{R}^3}\int_{R>0}\chi_{N\geq 1}\cdot(-2M)\frac{dz\,dR}{R^5}\\
&=-\frac{1}{\pi}\sum_k\int_{z\in\mathbb{R}^3}\int_{R>0}\chi_{N(z,R)\geq 1}\chi_{y_k\in B(z,R)}\frac{dz\,dR}{R^5}\\
&\geqslant-\frac{1}{\pi}\sum_k\int_{z\in\mathbb{R}^3}\int_{R>\delta(y_k)/2}\chi_{y_k\in B(z,R)}\frac{dz\,dR}{R^5}\\
&=-\frac{8}{3}\sum_k\frac{1}{\delta(y_k)}.
\end{aligned}$$

This is one expression of screening: Each nucleus feels (roughly) only the attraction of the nearest electron. The constant 8/3 here cannot be reduced below 3/2, since we can take a single nucleus midway between two electrons.

Returning to the proof of our main result, we use (2) and (6) to write

$$\begin{aligned}
\langle H\psi,\psi\rangle&\geqslant\left\langle\frac{1}{2\pi}\int_{z\in\mathbb{R}^3}\int_{R>0}\Omega\frac{dz\,dR}{R^5}\psi,\psi\right\rangle\\
&\quad+\sum_k\frac{3\varkappa}{16\pi^2}\int_{z\in\mathbb{R}^3}\int_{R>0}\int_{x'_k\in[{}^cB(z,R)]^{s-1}}T_k(z,R;x'_k)\,dx'_k\frac{dz\,dR}{R^8}\qquad(7)
\end{aligned}$$

$\equiv$ «Potential energy» term + «Kinetic Energy» term, where

$$\Omega=N(N-1)+M(M-1)-2MN+\varkappa(N-1)_+.\qquad(8)$$

Thus $\Omega = \Omega(z, R; x_1 \dots x_s)$.

In the sections to follow, we shall prove that the right-hand side of (7) is positive.

3. Allocating Energy Among the Nuclei

Note first that if $B(z, R)$ contains no nuclei then $M = 0$ and $\Omega \geqslant 0$. Hence for any fixed electrons $x_1 \dots x_s$ and nuclei $y_1 \dots y_{s'}$ we can write

$$V^+ \equiv \frac{1}{2\pi} \int_{z \in \mathbb{R}^3} \int_{R>0} \Omega \frac{dz\, dR}{R^5} \geqslant \sum_l \frac{1}{2\pi} \int_{z \in \mathbb{R}^3} \int_{R>0} \chi_{y_l \in B(z,R)} \frac{\Omega}{M} \frac{dz\, dR}{R^5}.$$

In other words, we allocate the «potential energy» Ω associated to $B(z, R)$ equally among the nuclei in $B(z, R)$.

Next note that $\Omega/M \geqslant e(\varkappa) > 0$ when $\varkappa > 4$ and $N \geqslant 2$; this is immediate from (8). Hence

$$V^+ \geqslant \sum_l \frac{1}{2\pi} \int_{\substack{z \in \mathbb{R}^3 \\ R>0}} [(M - 1 - 2N)\chi_{N \leq 1} + e(\varkappa)\chi_{N \geq 2}]\chi_{y_l \in B(z,R)} \frac{dz\, dR}{R^5}. \quad (9)$$

For fixed l, we shall make the change of variable $z = y_l + Rw$ in (9). In particular, $dz\, dR = R^3\, dw\, dR$, and $B(z, R) = B_l(w, R) \equiv B(y_l + Rw, R)$. We have $y_l \in B(z, R)$ if and only if $|w| < 1$. Define $R(k, l, w) = \sup\{R > 0 \mid B_l(w, R)$ contains no x_j with $j \neq k\}$ $B(k, l, w) = B_l(w, R)$ with $R = R(k, l, w)$.

Observe that $R(k, l, w)$ and $B(k, l, w)$ depend only on x'_k, not on x_k. Note also that $R \to B_l(w, R)$ is an increasing family for each fixed w ($|w| < 1$). Next, note that

$$\begin{bmatrix} \chi_{N(z,R) \geq 2} = \sum_k \chi_{R > R(k,l,w)} \chi_{B(k,l,w)}(x_k) \\ \chi_{N(z,R) \leq 1} = \sum_k \chi_{R < R(k,l,w)} \chi_{B(k,l,w)}(x_k) \end{bmatrix} \quad \begin{matrix} \text{for} \quad z = y_l + Rw, \\ |w| < 1. \end{matrix} \quad (10)$$

To check (10), we fix l, w ($|w| < 1$), and look at the union of the $B_l(w, R)$ containing at most one electron. This will have the form $B_l(w, R^*)$, and will contain exactly one electron. Here we ignore a set of w of measure zero. If that electron is x_k, then $R^* = R(k, l, w)$ and $\chi_{N(z,R) \leq 1} = \chi_{R < R^*}$ for $z = y_l + Rw$. So we obtain (10).

To proof of (10) shows also that $N = N(z, R) = \chi_{B_l(w,R)}(x_k)$ when $z = y_l + Rw$, $|w| < 1$, $N(z, R) \leqslant 1$ and $x_k \in B(k, l, w)$. Putting these remarks into (9), we obtain

$$V^+ \geqslant \frac{1}{\pi} \sum_{k,l} \chi_{B(k,l,w)}(x_k) \int_{|w|<1} \left\{ \frac{e(\varkappa)}{R(k, l, w)} + \int_0^{R(k,l,w)} \left[\frac{M-1}{2} - \chi_{B_l(w,R)}(x_k) \right] \frac{dR}{R^2} \right\} dw.$$

Integrating this against $|\psi(x_1 \dots x_s)|^2 dx_1 \dots dx_s$, we find that

$$\left\langle \frac{1}{2\pi} \int_{z\in\mathbb{R}^3} \int_{R>0} \Omega \frac{dz\, dR}{R^5} \psi, \psi \right\rangle \geqslant \frac{1}{\pi} \sum_{k,l} \int_{|w|<1} \int_{x'_k \in \mathbb{R}^{3s-3}} A(k, l, w, x'_k)\, dx'_k\, dw \tag{11}$$

where $A = A(k, l, w, x'_k)$ is given by

$$A = \int_{B(\bar{R})} |u(x)|^2 \left\{ \frac{e(\varkappa)}{2\bar{R}} + \int_0^R \left[\frac{M(R) - 1}{2} - \chi_{B(R)}(x) \right] \frac{dR}{R^2} \right\} dx \tag{12}$$

and for simplicity we have set

$$\bar{R} = R(k, l, w), \qquad B(R) = B_l(w, R), \qquad M(R) = \text{number of nuclei in } B(R),$$
$$u(x) = \psi(x, x'_k). \tag{13}$$

In particular,

$$B(R) \text{ is a ball of radius } R; \qquad R \to B(R) \text{ is increasing;} \qquad R \to M(R) \text{ is an increasing, positive integer-valued function.} \tag{14}$$

(Recall $y_l \in B_l(w, R)$ so $M(R) \geqslant 1$.)

Thus, the «potential energy» term on the right in (7) is built out of one electron energies of the form (12).

We next make an analogous study of the «kinetic energy» term. Again, we allocate the contribution of $B(z, R)$ equally among the nuclei contained in $B(z, R)$, discarding any balls with no nuclei inside. Thus,

$$T \equiv \sum_k \frac{3\varkappa}{16\pi^2} \int_{z\in\mathbb{R}^3} \int_{R>0} \int_{x'_k \in [{}^cB(z,R)]^{s-1}} T_k(z, R; x'_k)\, dx'_k \frac{dz\, dR}{R^8}$$
$$\geqslant \frac{3\varkappa}{16\pi^2} \sum_{k,l} \int_{z\in\mathbb{R}^3} \int_{R>0} \int_{x'_k \in [{}^cB(z,R)]^{s-1}} \frac{T_k(z, R; x'_k)}{M(z, R)} \chi_{y_l \in B(z,R)}\, dx'_k \frac{dz\, dR}{R^8}.$$

For each y_l we again use the change of variable $z = y_l + Rw$. As before, $y_l \in B(z, R)$ if and only if $|w| < 1$. Also $x'_k \in [{}^cB(z, R)]^{s-1}$ if and only if $R < R(k, l, w)$, the function of x'_k already defined above. Hence, in the notation (13), we have

$$\sum_k \frac{3\varkappa}{16\pi^2} \int_{z\in\mathbb{R}^3} \int_{R>0} \int_{x'_k \in [{}^cB(z,R)]^{s-1}} T_k(z, R; x'_k)\, dx'_k \frac{dz\, dR}{R^8}$$
$$\geqslant \frac{1}{\pi} \sum_{k,l} \int_{|w|<1} \int_{x'_k \in \mathbb{R}^{3s-3}} B(k, l, w, x'_k)\, dx'_k\, dw \tag{15}$$

with $B = B(k, l, w, x'_k)$ given by

$$B = \frac{3\varkappa}{16\pi} \int_0^{\bar{R}} \int_{x,y \in B(R)} |u(x) - u(y)|^2 \, dx \, dy \frac{dR}{R^5 M(R)}. \tag{16}$$

So the «kinetic energy» in (7) is also a superposition of one-electron energies.

Now, comparing (7), (11), (15), we see that $\langle H\psi, \psi \rangle \geqslant 0$ provided we can show that $A + \mathrm{B} \geqslant 0$ with A given by (12) and B by (16). This inequality involves a single electron and a single nucleus, but depends also on an unknown integer-valued function $R \to M(R)$. M has the bizarre effect of screening kinetic energy, while providing a constant term attributable to repulsion in the potential energy. In the next sections, we shall see how to simplify and prove $A + B \geqslant 0$.

4. Simplifying the One-Electron Problem

So far we reduced $\langle H\psi, \psi \rangle \geqslant 0$ to the following problem: Let $R \to B(R)$ be an increasing family of balls, with $B(R)$ having radius R. Let $R \to M(R)$ be an increasing, positive integer-valued function on $(0, \infty)$, and let $\bar{R} > 0$ be given. Prove that

$$\frac{3\varkappa}{16\pi} \int_0^{\bar{R}} \int_{x,y \in B(R)} |u(x) - u(y)|^2 \, dx \, dy \frac{dR}{R^5 M(R)}$$
$$+ \int_{B(\bar{R})} |u(x)|^2 \cdot \left\{ \frac{e(\varkappa)}{2\bar{R}} + \int_0^{\bar{R}} \left[\frac{M(R) - 1}{2} - \chi_{B(R)}(x) \right] \frac{dR}{R^2} \right\} dx \geqslant 0. \tag{17}$$

Now (17) is dilation-invariant, so we may take $\bar{R} = 1$. Moreover, we may assume that the $B(R)$ are all centered at 0, in view of the following simple result.

Lemma 2. *There is a measure-preserving transformation of* $\mathbb{R}^3$ *which carries* $B(R)$ *to* $B(0, R)$ *for all* R.

PROOF OF LEMMA 2. First, we make a preliminary change of variables Φ which carries $B(R)$ to $B(0, R)$; then we correct Φ to make it volume-preserving. To define Φ, let $\xi(R)$ be the center of $B(R)$, and pick a line thorough $\xi(R)$ parallel to $d\xi(R)/dR$. [If $(d\xi/dR)(R) = 0$, then any line through $\xi(R)$ will do, as long as its dependence on R is measurable.] Using this line as a polar axis, we pick in any measurable fashion a meridian on $\partial B(R)$, i.e., a great circle passing through the poles. The poles and the meridian determine latitude θ and longitude ϕ on $\partial B(R)$, and therefore give coordinates (R, θ, ϕ) for an arbitrary

point z of $\mathbb{R}^3$. Our map ϕ simply sends z to the point whose ordinary spherical coordinates are (R, θ, ϕ). Although ϕ needn't preserve volume, the new measure on $\mathbb{R}^3$ induced form ϕ will have the form $g(R, \theta) \cdot R^2\, dR \cos\theta\, d\theta\, d\phi$. Since $\Phi^{-1}(B(0, R)) = B(R)$ is a ball of radius R, we have

$$\int_{\theta=-\pi/2}^{\theta=\pi/2} 2\pi g(R, \theta) \cos\theta\, d\theta = 4\pi. \tag{18}$$

Now to make Φ volume-preserving, we just follow Φ with a change of coordinate $\psi: (R, \theta, \phi) \to (R, \tilde{\theta}, \phi)$ on $\mathbb{R}^3$ given by $\tilde{\theta}(-\pi/2) = -\pi/2$, $\cos\tilde{\theta}\, d\tilde{\theta} = g(R, \theta) \cos\theta\, d\theta$. In view of (18), we have $\tilde{\theta}(+\pi/2) = +\pi/2$, so our map is well-defined on $\mathbb{R}^3$ despite the amibiguities of longitude at the north and south poles. Now $\psi \circ \Phi$ carries $B(R)$ to $B(0, R)$ and preserves volume. □

We could have worked out an explicit formula in place of Lemma 2, since our $B(R) = B(y_l + Rw, R)$.

Returning now to (17), we may suppose $\bar{R} = 1$ and $B(R) = B(0, R)$, so our problem is to prove that

$$Q(u) = \frac{3\varkappa}{16\pi} \int_0^1 \int_{|x|,\,|y|<R} |u(x) - u(y)|^2\, dx\, dy \frac{dR}{R^5 M(R)} + \int_{|x|<1} \left(\epsilon - \frac{1}{|x|}\right) |u(x)|^2\, dx \geqslant 0 \tag{19}$$

with

$$\epsilon = 1 + \frac{e(\varkappa)}{2} + \int_0^1 \frac{M(R) - 1}{2} \frac{dR}{R^2}. \tag{20}$$

Up to now we worked with wave functions in C_0^∞. Since the transformation in Lemma 2 is merely measurable, we must prove (19) for all $u \in L^\infty$. Next, it is trivial to reduce (19) to the special case of radially symmetric u. In fact, for general u we just write $u = v + w$ with v radially symmtric and w having average zero on each sphere. We have

$$Q(u) = Q(v) + \left[\frac{3\varkappa}{16\pi} \int_0^1 \frac{8\pi}{3} R^3 \int_{|x|<R} |w(x)|^2\, dx \frac{dR}{R^5 M(R)} + \int_{|x|<1} \left(\epsilon - \frac{1}{|x|}\right) |w(x)|^2\, dx \right],$$

the point being that $Av_{|x|<R} w = 0$. Define a positive radial function $\tilde{w}$ by the condition $Av_{|x|=R} \tilde{w}^2 = Av_{|x|=R} |w|^2$, $R > 0$; then the expression in brackets evidently dominates $Q(\tilde{w})$. So (19) for radial u implies (19) in the general case.

So far, the increasing function $R \to M(R)$ is arbitrary. However, we observe that for $\varkappa \leqslant 12$, it is enough to prove (19) for functions $M(R)$ taking only the values $1, 2, 3$. In fact, suppose $M(R)$ takes the values $1, 2, 3, \ldots, M_f$ with $M_f \geqslant 4$ and the jump from $M_f - 1$ to M_f occurring at $R = R_f < 1$. Let us change $M(R)$ to $M'(R) = \min\{M(R), M_f - 1\}$, and see how Q changes in (19), (20). The first term in Q increases by

$$\frac{3\varkappa}{16\pi} \int_{R_f}^{1} \int_{|x|,\,|y|<R} |u(x) - u(y)|^2 \, dx\, dy \cdot \left(\frac{1}{M_f - 1} - \frac{1}{M_f}\right) \frac{dR}{R^5}$$

$$= \frac{3\varkappa}{16\pi M_f(M_f - 1)} \int_{R_f}^{1} \left(\frac{8\pi}{3} R^3 \int_{|x|<R} |u(x) - A v_{|y|<R} u(y)|^2 \, dx\right) \frac{dR}{R^5}$$

$$\leqslant \frac{\varkappa}{2(M_f - 1)M_f} \int_{R_f}^{1} \int_{|x|<R} |u(x)|^2 \, dx \frac{dR}{R^2}$$

$$\leqslant \frac{\varkappa}{2M_f(M_f - 1)} \left(\frac{1}{R_f} - 1\right) \|u\|^2.$$

On the other hand, ϵ drops by an amount

$$\frac{1}{2} \int_{R_f}^{1} \frac{dR}{R^2} = \frac{1}{2}\left(\frac{1}{R_f} - 1\right).$$

Hence the new quadratic form is at most equal to the old one plus

$$\left[\frac{\varkappa}{M_f(M_f - 1)} - 1\right] \cdot \frac{1}{2}\left(\frac{1}{R_f} - 1\right) \|u\|^2.$$

If $\varkappa \leqslant 12$ and $M_f \geqslant 4$, then the quantity in brackets is negative or zero. Hence $Q(u)$ decreases if we pass form $M(R)$ to $M'(R)$. Repeating this argument, we reduce (19) to the case $M(R) \leqslant 3$, all R. Since also $M(0) = 1$, the function $M(R)$ is now completely determined by the positions of its two jumps.

In the next section, we study the Euler-Lagrange equation associated to Q, and reduce (19) to a statement about ODE's.

5. Ordinary Differential Equations for the One Electrom Problem

We start off this section with some elementary estimates for quadratic forms related to (19).† First of all, for $\varphi(x)$ radial, square-integrable, and vanishing

† Throughout this section our basic Hilbert space is $L^2_{\text{radial}}(B(0, 1))$.

near 0 and ∞, we have the basic inequality

$$\frac{3}{2\pi}\int_0^\infty\int_{|x|,\,|y|<R}|\varphi(x)-\varphi(y)|^2\,dx\,dy\frac{dR}{R^5}-\int_{\mathbb{R}^3}|\varphi(x)|^2\frac{dx}{|x|}\geqslant 0. \tag{20a}$$

This may be verified by using the Mellin transform, that is, writing

$$\varphi(x)=\int_{-\infty}^{\infty}|x|^{-1+iY}[\tilde{\varphi}(Y)]\,dY.$$

The left side of (20*a*) then becomes

$$\int_{-\infty}^{\infty}m(Y)|\tilde{\varphi}(Y)|^2\,dY,$$

and one simply calculates $m(Y)$ and checks that it is non-negative.

Next, let $\psi(x)$ be radial and square-integrable on $|x|<1$, with $\psi(x)=0$ for $|x|\ll 1$. We set $\varphi(x)=\psi(x)$ for $|x|<1$, $\varphi(x)=0$ for $|x|\geqslant 1$, and apply (20*a*). Since

$$\int_{|x|,\,|y|<R}|\varphi(x)-\varphi(y)|^2\,dx\,dy\leqslant\frac{8\pi}{3}R^3\|\varphi\|^2=\frac{8\pi}{3}R^3\|\psi\|^2\quad\text{for}\quad R>1,$$

we obtain

$$\frac{3}{2\pi}\int_0^1\int_{|x|,\,|y|<R}|\psi(x)-\psi(y)|^2\,dx\,dy\frac{dR}{R^5}-\int_{|x|<1}|\psi(x)|^2\frac{dx}{|x|}\geqslant -4\|\psi\|^2. \tag{21}$$

Now fix $0<R_1\leqslant R_2\leqslant 1$, and define:

$$M(R)=\begin{bmatrix}1 & \text{for} & 0<R<R_1\\ 2 & \text{for} & R_1\leqslant R<R_2\\ 3 & \text{for} & R_2\leqslant R<1\end{bmatrix}$$

$$\mathcal{Q}(\psi)=\frac{3\varkappa}{16\pi}\int_0^1\int_{|x|,\,|y|<R}|\psi(x)-\psi(y)|^2\,dx\,dy\frac{dR}{R^5M(R)}-\int_{|x|<1}|\psi(x)|^2\frac{dx}{|x|} \tag{22}$$

for $\psi\in\mathfrak{D}_0=\{$square-integrable radial functions on $|x|<1$, vanishing near the origin$\}$.

Note that

$$\mathcal{Q}(\psi)=\frac{3\varkappa}{8\pi}\int_{|x|<1}\overline{\psi(x)}\left[\int_{|x|}^1\int_{|y|<R}(\psi(x)-\psi(y))\,dy\frac{dR}{R^5M(R)}\right]dx$$
$$-\int_{|x|<1}|\psi(x)|^2\frac{dx}{|x|}=\langle A\psi,\psi\rangle$$

for $\psi \in \mathfrak{D}_0$, with

$$A\psi(x) = W(x)\psi(x) - \int_{|y|<1} K(x,y)\psi(y)\,dy \tag{23}$$

$$W(x) = -\frac{1}{|x|} + \frac{3\varkappa}{8\pi}\int_{|x|}^{1}\int_{|y|<R} dy\,\frac{dR}{R^5 M(R)} \tag{24}$$

$$K(x,y) = \frac{3\varkappa}{8\pi}\int_{\max(|x|,|y|)}^{1} \frac{dR}{R^5 M(R)} \tag{25}$$

If $\varkappa \geqslant 8$, then comparison of (22), (21) shows that $\langle A\psi, \psi\rangle = \mathcal{Q}(\psi) \geqslant$ $\geqslant -C(R_1, R_2)\|\psi\|^2$ for $\psi \in \mathfrak{D}_0$.

Since also $\langle A\psi, \varphi\rangle = \langle \psi, A\varphi\rangle$ for $\psi, \varphi \in \mathfrak{D}_0$, we recall from basic functional analysis that A extends from $\mathfrak{D}_0$ to a self-adjoint operator $\bar{A}$, the Friedrichs extension of A. Specifically, $u \in$ Domain $(\bar{A})$ and $\bar{A}u = v$ if and only if we have $\langle u, A\psi\rangle = \langle v, \psi\rangle$ for $\psi \in \mathfrak{D}_0$, and there exist $\psi_k \in \mathfrak{D}_0$ converging to u in L^2-norm, with $\mathcal{Q}(\psi_k - \psi_l) \to 0$ as $k, l \to \infty$. These facts are contained in Dunford-Schwartz Vol. 2, Cor. 12.6.3.

It will be useful to have a concrete sufficient condition for a given radial $u \in L^2(|x| < 1)$ to belong to Domain $(\bar{A})$. We first note that

$$v(x) = W(x)u(x) - \int_{|y|<1} K(x,y)u(y)\,dy \tag{26}$$

is well-defined for all $u \in L^2$, because of the simple estimate $|K(x,y)| \leqslant$ $\leqslant C(|x| + |y|)^{-4}$. So if

$$\int_{|x|<1} |v(x)|^2\,dx < \infty \quad \text{and} \quad \int_{|x|<1} |u(x)|^2 \frac{dx}{|x|} < \infty \tag{27}$$

then $u \in$ Domain$(\bar{A})$, and $\bar{A}u = v$.

To see this, we first have to check that $\langle u, A\psi\rangle = \langle v, \psi\rangle$ for $\psi \in \mathfrak{D}_0$. That is easy, because the relevant integrals converge absolutely when $\psi \in \mathfrak{D}_0$, and hence their order can be switched. We much also find $\psi_k \to u$ in L^2 with $\mathcal{Q}(\psi_k - \psi_l) \to 0$. It is enough to take $\psi_k(x) = u(x)\chi_{|x|>1/k}$, and note that

$$\mathfrak{D}(\psi_k - \psi_l) \leqslant C\int_{|x|<\max[(1/k),(1/l)]} |u(x)|^2 \frac{dx}{|x|} \to 0$$

if (27) holds. Thus we have a sufficient condition (27) for $u \in$ Domain $(\bar{A})$, and a formula (26) for $\bar{A}u$.

The main point of this section is to reduce questions about the spectrum of $\bar{A}$ to a study of ordinary differential equations.

From now on, we fix $\varkappa = 12$. Let $R_1 = e^{t_1}$, $R_2 = e^{t_2}$, so that $t_1 < t_2 < 0$. Fix $\epsilon > 1$, not necessarily given by (20).

The relevant differential equations turn out to be as follows.
For $t_2 \leqslant t \leqslant 0$ we solve

$$\frac{d^2\mathfrak{F}}{dt^2} + 4\frac{d\mathfrak{F}}{dt} + \frac{6}{1 + [\epsilon - 2]e^t}\mathfrak{F} = 0 \tag{28}$$

with boundary conditions

$$\mathfrak{F}(0) = 0 \qquad \frac{d\mathfrak{F}}{dt}(0) = 1. \tag{29}$$

Having found $\mathfrak{F}$, $d\mathfrak{F}/dt$ at $t = t_2$, we next solve

$$\frac{d^2\mathfrak{G}}{dt^2} + 4\frac{d\mathfrak{G}}{dt} + \left(\frac{9}{2 + [\epsilon - 2 - e^{-t_2}]e^t}\right)\mathfrak{G} = 0 \tag{30}$$

in the region $t_1 \leqslant t \leqslant t_2$, with boundary conditions

$$\mathfrak{G}(t_2) = \mathfrak{F}(t_2), \qquad 2\frac{d\mathfrak{G}}{dt}(t_2) = 3\frac{d\mathfrak{F}}{dt}(t_2). \tag{31}$$

Next we solve

$$\frac{d^2\mathfrak{K}}{dt^2} + 4\frac{d\mathfrak{K}}{dt} + \left(\frac{18}{5 + [\epsilon - 2 - e^{-t_2} - 3e^{-t_1}]e^t}\right)\mathfrak{K} = 0 \tag{32}$$

in the region $-\infty < t \leqslant t_1$, with boundary condition

$$\mathfrak{K}(t) \sim \exp\left(\left[-2 + \sqrt{\frac{2}{5}}\right]t\right) \quad \text{as} \quad t \to -\infty. \tag{33}$$

Then we say that a *match* occurs for ϵ, t_1, t_2 if the vectors

$$\left(\mathfrak{K}(t_1), \frac{d\mathfrak{K}}{dt}(t_1)\right) \quad \text{and} \quad \left(\mathfrak{G}(t_1), 2\frac{d\mathfrak{G}}{dt}(t_1)\right)$$

are proportional. Define the *discriminant* for ϵ, t_1, t_2 as

$$\left[2\frac{d\mathfrak{G}}{dt}(t_1)\cdot\mathfrak{K}(t_1) - \mathfrak{G}(t_1)\frac{d\mathfrak{K}}{dt}(t_1)\right] \Big/ \left(|\mathfrak{K}(t_1)| + \left|\frac{d\mathfrak{K}}{dt}(t_1)\right|\right)\cdot$$
$$\cdot\left(|\mathfrak{G}(t_1)| + \left|\frac{d\mathfrak{G}}{dt}(t_1)\right|\right).$$

Lemma 3. *For $\epsilon > 1$, we have $-\epsilon \in \text{Spectrum}(\bar{A})$ if and only if a match occurs.*

PROOF. Let us study the equation $(\epsilon + \bar{A})u = f$ for radial $f \in C_0^\infty(|x| < 1)$ vanishing near the origin. According to (26), (27), it will be enough to solve

$$(\epsilon + W(r))u(r) - \int_0^1 K(r, s)4\pi s^2 u(s)\, ds = f(r) \tag{34}$$

with

$$\int_0^1 \frac{|u(r)|^2}{r} \cdot 4\pi r^2\, dr < \infty,$$

u continuous away from 0. (Finally we have written radial u on $\mathbb{R}^3$ as a function of one variable.)

Now

$$K(r, s) = H(\max\{r, s\}) \quad \text{with} \quad H'(r) = -\frac{3\varkappa}{8\pi} \cdot \frac{1}{r^5 M(r)}.$$

So, differentiating (34) in r, we get

$$\frac{d}{dr}[(\epsilon + W)u] + \frac{3\varkappa}{8\pi r^5 M(r)} \int_0^r 4\pi s^2 u(s)\, ds = \frac{d}{dr} f. \tag{35}$$

Also $(\epsilon + W)u(l) = 0$ by (34), since $f(1) = 0$ and $K(1, s) = 0$ for $s < 1$.

Conversely, $(\epsilon + W)u(1) = 0$ and (35) yields (34) by integration. So we must solve (35) with the boundary conditions: $(\epsilon + W)u(1) = 0$, $\int_0^1 r|u(r)|^2\, dr < \infty$, u continuous away from 0.

Next multiply (35) by $r^5 M(r)$ and differentiate away from the jumps of $M(r)$. The result is

$$M(r)\frac{d}{dr} r^5 \frac{d}{dr}[(\epsilon + W)u] + \frac{3\varkappa}{2} r^2 u(r) = M(r)\frac{d}{dr} r^5 \frac{d}{dr} f \tag{36}$$

away from the jumps of $M(r)$. Also from (35), we see that

$$M(r)\frac{d}{dr}[(\epsilon + W)u - f]$$

is continuous across the jumps of $M(r)$. Conversely, if

$$M(r)\frac{d}{dr}[(\epsilon + W)u - f]$$

is continuous across the jumps of $M(r)$, then we may integrate (36) to obtain

$$r^5 M(r)\frac{d}{dr}[(\epsilon + W)u] + \frac{3\varkappa}{8\pi} \int_0^r 4\pi s^2 u(s)\, ds = r^5 M(r)\frac{d}{dr} f + (\text{Const.}).$$

If as $r \to 0$ we have $u(r) = O(r^{-\alpha})$ and $(du/dr) = O(r^{-\alpha-1})$, with $\alpha < 1$, then the Constant term must vanish, and we recover (35). Also $\int_0^1 r|u(r)|^2\,dr < \infty$ if $u = O(r^{-\alpha})$. So $(\epsilon + \bar{A})u = f$, provided (36) holds, and:

$$(\epsilon + W)u(1) = 0,\ u \text{ continuous on } (0,1),\ M(r)\frac{d}{dr}[(\epsilon + W)u - f]$$
$$\text{continuous across the jumps of } M(r);\ u(r),\ ru'(r) = O(r^{\delta-1}) \text{ as } r \to 0. \tag{37}$$

Now we can check whether $-\epsilon \in \mathrm{Spec}(\bar{A})$. If a match occurs, then we can solve (36), (37) with $f = 0$. In fact, taking $r = e^t$ and $(\epsilon + W)u(r) = \mathfrak{F}, \mathfrak{G}, \mathfrak{H}$ in the three regions $[t_2, 0]$, $[t_1, t_2]$, $(-\infty, t_1)$, we find that (28)...(33) with a match amount to (36), (37). We use $\epsilon > 1$ because then $\epsilon + W \neq 0$ everywhere. Thus $\epsilon + \bar{A}$ has nontrivial kernel, and $-\epsilon \in \mathrm{Spec}(\bar{A})$. Conversely, suppose a match doesn't take place at t_1, t_2. For $f \in C_0^\infty(0,1)$ we can then solve (36), (37) by means of a Green's function. Setting $F = (\epsilon + W)u - f$, we rewrite (36), (37) in the form

$$M(r)\frac{d}{dr}r^5\frac{d}{dr}F + \frac{3\varkappa}{2}\left(\frac{r^2}{\epsilon + W}\right)F = -\frac{3\varkappa}{2}\left(\frac{r^2}{\epsilon + W}\right)f \tag{38}$$

$$F(r), \qquad M(r)\frac{dF}{dr} \quad \text{continuous across the jumps of } M(r) \tag{39}$$

$$F(1) = 0 \tag{40}$$

$$F(r) = O(r^{-2+\delta}), \qquad F'(r) = O(r^{-3+\delta}) \quad \text{as} \quad r \to 0. \tag{41}$$

To find a Green's a function for (38)...(41), we first set $f = 0$ in (38), and pick out two special solutions of the homogeneous equation: Set $F_+(r)$ = solution of (38), (39), (40); set $F_-(r)$ = solution of (38), (39), (41). Since a match does not occur, these solutions are distinct. Our Green's function takes the form

$$K(r,s) = \begin{bmatrix} C_1(s)F_+(r) & \text{if} \quad r > s \\ C_2(s)F_-(r) & \text{if} \quad r \leqslant s \end{bmatrix}$$

where $C_1(s)$, $C_2(s)$ are picked so that $r \to K(r,s)$ is continuous at $r = s$, while

$$r \to \frac{\partial K(r,s)}{\partial r.} \quad \text{jumps by an amount} \quad \frac{1}{M(s)s^5} \quad \text{at} \quad r = s$$

As $_r \to 0$, we have $F_-(r) \sim r^{-2+\sqrt{2/5}}$, $F_+(r) \sim r^{-2-\sqrt{2/5}}$. Consequently, we may estimate $C_1(s)$, $C_2(s)$ as $s \to 0$; in fact $|C_1(s)| \leqslant Cs^{\sqrt{2/5}}$, $|C_2(s)| \leqslant Cs^{-\sqrt{2/5}}$

for $0 < s \leqslant 1$, so

$$|K(r,s)| \leqslant Cr^{-2}\left[\min\left(\frac{r}{s},\frac{s}{r}\right)\right]^{\sqrt{2/5}}. \tag{42}$$

Equations (38)...(41) are now solved by taking

$$F(r) = \int_0^1 \frac{3\varkappa}{2}\left(\frac{s^2}{\epsilon + W(s)}\right) f(s)\cdot K(r,r)\,ds, \tag{43}$$

$$u = \frac{F+f}{\epsilon + W}. \tag{44}$$

Having solved $(\epsilon + \bar{A})u = f$, we next note the a-priori bound

$$\|u\|_{L^2(|x|<1)} \leqslant C\|f\|_{L^2(|x|<1)}. \tag{45}$$

In fact, from (42), (43), (44), we have

$$|u(r)| \leqslant CTf(r) = C\int_0^\infty \left[\mathrm{MIN}\left(\frac{r}{s},\frac{s}{r}\right)\right]^{\sqrt{2/5}} f(s)\frac{ds}{s},$$

and (45) follows at once by the Mellin transform.

Now if ϵ is any number, $\bar{A}$ is any self-adjoint operator, and for f in a dense subspace we can solve $(\epsilon + \bar{A})u = f$ with a bound $\|u\| \leqslant C\|f\|$, then $-\epsilon \notin$ Spectrum $(\bar{A})$, as follows at once from the spectral theorem. Hence in the context of Lemma 3 we have $-\epsilon \notin$ Spectrum $(\bar{A})$ if a match does not occur. □

We can now state what we need about ODE's in order to prove our main result, namely

Lemma 4. *No match occurs in any of the following ranges of* t_1, t_2, ϵ:

(*a*) $t_1 = t_2 = 0, \qquad \frac{9}{4} \leqslant \epsilon \leqslant 4,$

(*b*) $-2 \leqslant t_1 \leqslant t_2 \leqslant 0, \qquad \epsilon = \frac{5}{4} + \frac{1}{2}(e^{-t_1} + e^{-t_2}),$

(*c*) $t_1 = -2, \qquad -2 \leqslant t_2 \leqslant 0, \qquad \frac{1}{2}(e^{-t_1} + e^{-t_2}) \leqslant \epsilon \leqslant \frac{5}{4} + \frac{1}{2}(e^{-t_1} + e^{-t_2}),$

(*d*) $t_1 = -2, \qquad t_2 = 0, \qquad \frac{e^2}{2} \leqslant \epsilon \leqslant \frac{e^2}{2} + \frac{7}{4}.$

We check here that Lemma 4 yields (19), (20) and thus proves our main result. In the sections to follow, we give a computer-assisted proof of Lemma 4. Define $\lambda_1(t_1, t_2) = \inf \text{Spectrum}(\bar{A})$ with $R_1 = e^{t_1}$, $R_2 = e^{t_2}$; and set $\epsilon(t_1, t_2) = = 5/4 + (e^{-t_1} + e^{-t_2})/2$. Note that $\lambda_1(t_1, t_2)$ varies continuously in t_1, t_2, since a small change in t_1, t_2 will change $\bar{A}$ by adding an operator of small norm. We shall prove (19), (20) by looking at the four cases

(α) $R_1 = R_2 = 1$,
(β) $e^{-2} \leqslant R_1 \leqslant R_2 \leqslant 1$,
(γ) $R_1 < e^{-2}$, $\quad R_1 \leqslant R_2 \leqslant e^2 R_1$,
(δ) $e^2 R_1 < R_2 \leqslant 1$.

Note that (19), (20) with $R_1 = e^{t_1}$, $R_2 = e^{t_2}$ is equivalent to $\lambda_1(t_1, t_2) \geqslant \geqslant -\epsilon(t_1, t_2)$.

Take case (α) first. Estimate (21) gives $\lambda_1(0, 0) \geqslant -4$, while Lemmas 3 and 4($a$) show that $\lambda_1(0, 0)$ cannot be in $[-4, -\epsilon(0, 0)]$. Hence $\lambda_1(0, 0) > -\epsilon(0, 0)$, and case ($\alpha$) is settled. Now suppose (19), (20) are violated for some R_1, R_2 in case (β). Then we must have $\lambda_1(t_1, t_2) = -\epsilon(t_1, t_2)$ for some t_1, t_2 with $-2 \leqslant t_1 \leqslant t_2 \leqslant 0$. (This is required in view of case (α) and the continuity of λ_1, ϵ.) On the other hand, Lemmas 3 and 4(b) imply $\lambda_1(t_1, t_2) \neq -\epsilon(t_1, t_2)$ for $-2 \leqslant t_1 \leqslant t_2 \leqslant 0$. So case ($\beta$) is settled also.

Next, by combining case (β) with Lemmas 3 and 4(c), we get

$$\frac{3\varkappa}{16\pi} \int_0^1 \int_{|x|, |y| < R} |u(x) - u(y)|^2 \, dx \, dy \frac{dR}{R^5 M(R)} - \int_{|x| < 1} |u(x)|^2 \frac{dx}{|x|} + \left(\frac{1}{2R_1} + \frac{1}{2R_2} \right) \|u\|^2 \geqslant 0 \qquad (46)$$

where

$$M(R) = \begin{bmatrix} 1 & \text{for} & R & < R_1 = e^{-2} \\ 2 & \text{for} & R_1 \leqslant R & < R_2 \\ 3 & \text{for} & R_2 \leqslant R & < 1 \end{bmatrix} \quad \text{and} \quad e^{-2} \leqslant R_2 \leqslant 1.$$

Similarly, using case (β) with Lemmas 3 and 4(d) yields

$$\frac{3\varkappa}{16\pi} \int_0^1 \int_{|x|, |y| < R} |u(x) - u(y)|^2 \, dx \, dy \frac{dR}{R^5 M(R)} - \int_{|x| < 1} |u(x)|^2 \frac{dx}{|x|} + \frac{1}{2R_1} \|u\|^2 \geqslant 0 \qquad (47)$$

where

$$M(R) = \begin{bmatrix} 1 & \text{for} & R & < R_1 = e^{-2} \\ 2 & \text{for} & R_1 \leqslant R & \leqslant 1 \end{bmatrix}.$$

Inequalities (46), (47) will take care of cases (γ), (δ). In fact, to handle (γ), set $R_* = e^2 R_1$ and rescale (46):

$$\frac{3\varkappa}{16\pi}\int_0^{R_*}\int_{|x|,\,|y|<R}|u(x)-u(y)|^2\,dx\,dy\frac{dR}{R^5M(R)}-\int_{|x|<R_*}|u(x)|^2\frac{dx}{|x|}+\left(\frac{1}{2R_1}+\frac{1}{2R_2}\right)\int_{|x|<R_*}|u(x)|^2\,dx\geqslant 0.$$

Adding to this the obvious inequalities

$$-\int_{|x|>R_*}|u(x)|^2\frac{dx}{|x|}+\left[\frac{1}{2R_1}+\frac{1}{2R_2}\right]\int_{|x|>R_*}|u(x)|^2\,dx\geqslant 0$$

$$\frac{3\varkappa}{16\pi}\int_{R_*}^{1}\int_{|x|,\,|y|<R}|u(x)-u(y)|^2\,dx\,dy\frac{dR}{R^5M(R)}+\frac{e(\varkappa)}{2}\|u\|^2\geqslant 0,$$

we obtain (19), (20) in case (γ). Finally, in case (δ), we rescale (47) to get:

$$\frac{3\varkappa}{16\pi}\int_0^{R_*}\int_{|x|,\,|y|<R}|u(x)-u(y)|^2\,dx\,dy\frac{dR}{R^5M(R)}-\int_{|x|<R_*}|u(x)|^2\frac{dx}{|x|}+\frac{1}{2R_1}\int_{|x|<R_*}|u(x)|^2\,dx\geqslant 0.$$

Adding to this the obvious inequalities

$$-\int_{|x|>R_*}|u(x)|^2\frac{dx}{|x|}+\frac{1}{2R_1}\int_{|x|>R_*}|u(x)|^2\,dx\geqslant 0$$

$$\frac{3\varkappa}{16\pi}\int_{R_*}^{1}\int_{|x|,\,|y|<R}|u(x)-u(y)|^2\,dx\,dy\frac{dR}{R^5M(R)}+\left(\frac{1}{2R_2}+\frac{e(\varkappa)}{2}\right)\|u\|^2\geqslant 0,$$

we obtain (19), (20) in case (δ).

Thus, our main result is reduced to Lemma 4. Since we took $\varkappa = 12$, our main result takes the form

$$\sum_k(-\Delta_{x_k})^{1/2}+\alpha\left[\sum_{j<k}\frac{1}{|x_j-x_k|}+\sum_{j<k}\frac{1}{|y_j-y_k|}-\sum_{j,k}\frac{1}{|x_j-y_k|}\right]\geqslant 0$$

with

$$\alpha=\frac{2}{\pi}\cdot\frac{35}{144}\sim\frac{1}{2.06\pi}.$$

6. Rigorous computer solution of ordinary differential equations

In this section, we explain in detail how Lemma 4 was actually established. In broad terms, what we did is to reduce the statements to be checked to a finite computation (expand the relevant quantities in appropriate parameters and bound the remainders) and then, perform this finite computation with the help of an electronic device (a VAX 11/750).

The fact that even finite arithmetic operations can be performed rigorously by a computer is not obvious (and indeed somewhat surprising for many of today's mathematicians).

The most serious reason is that computers are not equipped with real arithmetic, but only with some caricature defined on a finite set of numbers (those the machine can understand, usually called machine numbers) and yielding only approximate answers. The difficulty is that when approximations are taken a large number of times (e.g. the number of operations peformed by a computer in a minute) the result may not be a reasonable approximation any more. (For example, most computers are unable to guarantee even the sign of $f^{100}(\sqrt{3}/2)$ where $f(x) = 1 - 2x^2$.)

The cure for this difficulty is to work systematically with upper and lower bounds of numbers rather than with approximations to them, and to implement an arithmetic working on bounds (e.g. the addition for upper bounds should produce an upper bound to the true sum of the two original upper bounds).

That is, we work with intervals, and construct operations that given two intervals produce a third. That third interval contains the result of the true operation when the arguments range in the original intervals.

In other words, we build an arithmetic that takes into account roundoff errors. Then we will perform some analysis to reduce the original problem to an arithmetic one; that is, we take into account both roundoff and truncation errors.

The introduction of interval arithmetic can be attributed to R.E. Moore in 1962 and has a developed literature (see [Mo] [KM] and references there). The application to mathemtical problems was fostered by the pioneering work of Lanford [La] who found a quite remarkable abstract framework in which several hitherto unapproachable problems could be reduced to arithmetic statements, and wrote a package well adapted to those needs. Under his influence, similiar strategies have been applied for several other problems [E, K, *W*], [E, W] [MP] [LL] [Me].

In all the references above (except in [MP]) interval arithmetic was used to provide rigorous error bounds for certain computations; here we want to show that certain conditions do not happen for parameters ranging in a cer-

tain region. Even if the basic operations are the same, we are forced to deal with intervals much wider than the roundoff unit, and some considerations, safely ignorable for sharp intervals, come into the foreground (e.g. coherence and subdistributivity). Several features —implemented in our package— desirable for the handling of sharp intervals (e.g. double precision, 0 and 1 being neutral for addition and multiplication) become irrelevant when the intervals are «fat» and presumably take a toll in execution time, although, clearly, not in correctness. We preferred not to revise for speed, since in the present form the package had alredy been checked.

In a first subsection, we explain how the arithmetic package was implemented in our case.

In a second subsection, we document the reduction of the original question to an arithmetic problem. We refer the reader to [T] for computer concepts we use without explanation.

A. Interval arithmetic

Our implementation of interval arithmetic was done using the facilities provided by the manufacturer. The alternative of writing from scratch the arithmetic subroutines not only would have required more programming effort but also have resulted in programs running between 10 to 100 times slower.

The most authoritative source about what the VAX can do is [DEC]. For the purposes of this section, this could be the final description of our computer.

The only objects we are concerned about are «machine integers» and «machine reals» which are respectively

$$\mathcal{I} = \{(S, I)|, S = +1, -1, I \in N, -2^{31} \leqslant I < 2^{31}\}$$
$$\mathcal{R} = \{(S, e, m) \mid S = -1, +1; 0 \leqslant e \leqslant 2^9 - 1\}$$
$$0 \leqslant m \leqslant 2^{56} - 1\}$$

When $e \neq 0$, $(S; e, m)$ is supposed to mean

$$S \cdot 2^{(e-128)}\left(\frac{m + 2^{56}}{2^{57}}\right).$$

When $e = 0$, and $S = +1$, (S, e, m), is supposed to mean zero regardless of what m is.

When $e = 0$ and $s = -1$, this number is not supposed to mean anything and according to the manual, appearance of this number could cause unpredictable results [DEC] p. 34. The machine can understand and manipulate other numbers, buy they do not appear in our programs.

We will also be reading and writing numbers, which involves switching from the internal representation of the machine to another more familiar to the users. Since, for our problem, reading and writing was only a heuristic aid, we did not care about writing careful input and output subroutines. (This was done e.g. in [LL].)

On machine integers and reals we will be performing several operations: arithmetic operations, comparisons and conversions from integer to real.

We will assume —and we checked in several crucial cases— that the comparisons and type conversions worked correctly. A moment's reflection will show that there is no hope that the arithmetic operations work as those on real numbers.

Arithmetic operations on integers are performed mod 2^{32} and, with this proviso, they are exact. Real operations are more complicated. In [DEC] appendix H, it is stated that, when both the oeprands and the result are in $\mathfrak{R}$, the result is correct to «1/2 of the least significant bit.»

When the result is not in the range of acceptable numbers, the computer reports this condition but goes ahead. (These «exceptions» are used by the compilers and we will explain how the ones we used dealt with them.) This leads to a «circular arithmetic» in integers —arithmetic modulo 2^N, with the fundamental domain depending on which way the numbers are represented [T] p. 447 ($N = 32$ for the VAX 11/750 and notation of 2's complement)— and quite unpredictable results when there is overflow in floating point numbers.

With this information at hand, it is clear what we should do to obtain a controlled arithmetic: we should watch out for the signals that report an improper operation and modify the result in ways suitable for the arithmetic of upper and lower bounds. The «1/2 of least significant bit» accuracy implies that, any function right: $\mathfrak{R} \to \mathfrak{R}$ satisfying

$$\text{right}\ (x) > x$$

will also have the property

$$\text{right}\ (x \&_c y) > x \& y$$

where $\&$ and $\&_c$ stand respectively for any of the four arithmetic operations and its computer implementation. This holds (provided the operations has a result in $\mathfrak{R}$). Similarly for an operation left satisfying $\text{left}\ (x) < x$.

Even if any such operations left, right are acceptable, it is clear that it would be to our advantage to make them as small modifications as possible. The best we could do is just modify the least significant bit.

The programs considered here will be written in a high level language that will eventually get translated into machine language. Usually, this translation

will include handling of the signals of incorrect operations. The most usual convention for modern compilers —we checked that it was the case for the compilers we used— is as follows: let the circular arithmetic of the integers go unnoticed: when the result of an operation in $\mathfrak{R}$ has too small absolute value, set the result to zero. When the absolute value is too big, or there is a division by zero, print error messages and terminate the program. (In our system, the C-library included facilities to change the handling of those exceptions, but the standard ones were adequate.)

With this way of handling the exceptions, the only thing we have to worry about is zero which many appear as the result of an underflow. A moment's reflection will show that rounding to any number strictly different from zero is still adequate; but we should remember that, for the internal representation, this means modifying e and not m.

Adequate versions of right and left could be the ones listed at the end where one plus, one minus, zero plus and zero minus are global variables. Notice that their value does not matter too much, because at the end we check the strict inequality of the return value of the argument and as we pointed out, by the «1/2 LSB» property of the VAX operation this suffices.

Notice that, even in the case that one plus and one minus did not produce a sufficient effect, the only thing that would happen is termination of the program and not propagation of the wrong result.

We also used another version of these routines that only modifies the last bit. Unfortunately, since Pascal does not have bitwise manipulations this version was written in Fortran 77 and then grafted into the Pascal programs, a rather delicate task.

From the strictly logical point of view the first version is enough, but the second one also provided some insight into the workings of the machine.

The main idea is to use Fortran integer arithmetic to access the bits of a double precision number. This is achieved by using the equivalence statement between an array of two byte integers and a copy of the double precision argument. Double precision numbers are stored in four consecutive words whose significance decreases with increasing address (the first one contains the sign and exponent). Inside each word, the significance increases with decreasing address [DEC].

Integers are stored in a word with increasing significance given to decreasing addresses. Moreover, they are stored in two's complement notation with bit 15 designating the sign.

To increase the last bit of m, we increase the least significant bit of the last element of the array and check if this produced a carry. If it did not, we are satisfied, but if it did, we increment the next one. (Here, the fact that the integers are represented as two's complement is used in two places: one that increasing by 1 as a signed quantity is the same as an unsigned quantity and,

second that the carry in unsigned addition is produced exactly when the result —intepreted in two's complement is zero.)

This process of incrementing and carrying, if necessary, is repeated without any change till the first word where we should worry about the extra structure of exponent and sign. By a truly miraculous coincidence it turns out that the same prescription continues to work in the first word. We can think of the mantissa as representing .1m, where the leading 1 is not explicitly written. If by adding 1, the mantissa overflows, then the incremented mantissa is 1.00...0 whose internal representation would be achieved by incrementing the exponent by 1 and setting the mantissa to zero. This is what happens when we increment the first word as an unsigned bit pattern.

The only thing to do is to check that this carry did not affect the sign bit. If it did, we stop.

The exceptional case of zero is treated at the beginning and what we do is to set a bit pattern which corresponds to $e = 1$, $m = 0$, $s = +1$.

Similar considerations apply to the routine down moving the number towards zero. The only difference is that we set to zero all the numbers that, when divided by 4 underflow. Even if wasteful, this is expedient and certainly correct (because of the 1/2 LSB property).

Thus we have procedures «up», «down» which increase or decrease slightly the absolute value of a machine number. Once we have «up», «down», writing «right» and «left» is very easy.

Other versions acieving the same were also written for the VAS 11/75 in C by *D.* Rana on other principles, and there are versions working for other computers described in the references. Translating our Fortran routines to C is very easy using type casts rather than equivalence to assign double precision to integer arrays. Since quite a lot of system programming is done in *C*, the compilers tend to be well taken care of and, since the language is better designed than Fortran, it is harder to make mistakes writing the compiler. Moreover, in our environment, it turned out that grafting C was easier than grafting Fortran. Even if these considerations are quite machine dependent (and surely are reversed in other machines), we wanted to make the reader aware of this difficulty.

Once we have this rounding constructing an arithmetic package is reasonably straighforward.

We introduce three new structures which are bound, interval, complex: consisting respectively of a real field .top, supposedly a positive upper bound; two real fields .upper and .lower upper and lower bounds for the interval; two interval fields .re, .im which are intervals containing the real and imaginary part.

This package is really more than what the problem in its present version needs —for example, we do not make any use of complex numbers— but we

had used it for other type of problems —in fact, many of the names come from [LL] where the basic operations are implemented in a very different way.

We hope that the names of the subroutines are rather self explanatory and the working straightforward. Usually the name has some remainder of the function and a few extra letters remingding the reader of the types on which it acts (i:integer, v:interval, b:bound, c:complex).

We have tried to enforce the compatibility with standard Pascal and, with two exceptions, included mainly for experimenting, have avoided functions returning structures —which is de facto standard even though forbidden in the original description of the language [J.W.].

Even if the type conversions and comparisons were done using only those the compiler provided, we have included comparison and conversions as explicit subroutines with the hope of having a clean interface with the machine. If only those we supply are used, changes in the internal representation should only affect this level and not the upper levels. (We sometimes slackened and used compiler comparisons but they could easily be removed.) Notice that, when we want to assign an integer value to a real, we first assign it to an integer variable and, then, the integer to a real. This produces something strictly correct whereas writing x: = 2.0; would have made the computer involve the routine to convert decimals to internal, which gets the last bits wrong.

In particular, this interface was implemented by another different method in [LL]. Changing the programs there to run in the VAX was a question of a few hours of works. To adapt the present programs to the IBM PC would take only slightly more; this could be interesting because, for those susbstantial programs we run on both machines, the speeds were comparable in CPU time and the small computer was superior in real time since the big one has to be shared.

One important point of this package is the variable «status,» which is initialized to 0 at the beginning and, will take the value of 1 if some division by an interval containing 0 is requested.

The reason why we did that rather than killing the program from the division subroutine is, again, to allow different implementations of it.

One important part of checking the results is to check that this «sticky bit» is still zero.

We also had to write some subroutines for special functions —the only one used is the interval valued exponential of real intervals

$$\text{expx} = 1 + \frac{x}{1}\left(1 + \frac{x}{2}\left(1 + \frac{x}{3}\cdots\right)\right)$$

We select an N and then, produce an interval that contains

$$R \equiv \left(1 + \frac{x}{N+1}\left(1 + \frac{x}{N+2}\cdots\right)\right) \quad \text{when } x \text{ ranges in the input}$$

interval and then evaluate the product with interval arithmetic. We use the crude bounds

$$1 \geqslant R \geqslant 1 + \frac{x}{N+1} \quad \text{if} \quad -(N+1) < x < 0$$

$$1 \leqslant R \leqslant \frac{1}{1 - \frac{x}{N+1}} \quad \text{if} \quad N+1 > x > 0.$$

This procedure, even if correct in the interval $(-(N+1), N+1)$ is quite inefficient. What we do is to use this only when the input interval is in a small interval (about $(-1/8, 1 \times 8)$) and reduce to this case by using $e^x = (e^{x/2})^2$ as many times as needed; using recursion —allowed in Pascal— this is quite easy to program. At the beginning, we also check that the interval is within reasonable bounds. Even if this was not strictly necessary, the exponential ceases to work well for intervals not within reasonable bounds.

We also include routines to compute sin, cos for reals and exponential for complex. It is amusing to note that, for complex exponential it is much more accurate for fat intervals to use Euler's formula and call the real subroutines that to use recursion involving complex numbers.

Since we will very often be computing $e^x - 1$ for small x we included a special subroutine that does not reqire to subtract two very similar numbers. It is also based in using Taylor expansion grouped in a way similar to the exponential and the recursion relation $(e^x - 1) = (e^{x/2} - 1)(e^{x/2} + 1)$. Those subroutines are included in cexpo.i and a test program is testexpo.p.

We also have a right computation of powers which works on positive numbers and rounds to the right, and a left one. Out of those it is very simple to compute powers for intervals and a root; the idea for the latter is to compute the root approximately using the ones supplied by the compiler and then round appropriately until we can show using the previously mentioned powers that the power of this interval contains (not is contained in!) the given interval.

B. Excluding matches

In this subsection we describe how to compute bounds for discriminants with finitely many arithmetic operations (that is, we take into account truncation errors) and we will discuss how to use that to show that there are no matches in the regions required by Lemma 4.

One can observe that all the equations to be studied are of the form

$$D^2\psi + 4D\psi + \frac{\alpha}{1 + \epsilon e^t}\psi = 0$$

for suitable choices of α, ϵ, and it will suffice to solve two problems associated to this equation.

I) Given the asymptotic behavior as $t \to -\infty$ find the ratio of $\psi(t_0)$, $\psi'(t_0)$ at some negative t_0.

II) Given ψ and ψ' at some point t_0 find ψ and ψ' at another point t_1.

If our algorithms are going to be useful for present purposes, they should be able to work with intervals. That is, given that we know bounds for ϵ, we should be able to produce upper and lower bounds for the results.

Problem II will be solved in the most conservative way possible. That is: given intervals $t_0, t_1 \psi(t_0), \psi'(t_0)$. We will produce intervals $\psi(t_1)$ and $\psi'(t_1)$ in such a way that any initial data contained in $\psi(t_0)$, $\psi'(t_0)$ for an initial time contained in t_0, will be contained in $\psi(t_1)$ $\psi'(t_1)$ for any final time contained in t_1. (Of course, we will obtain estimates uniform in ϵ ranging in a given interval.)

Even if less conservative solutions would also be useful, this hast the advantage that it can be iterated quite straightforwardly. The iteration can be made without terrible losses in accuracy, as we will explain.

Rather than implementing a general O.D.E. solver, we have drawn freely on specific properties of the equation we were strudying. Even if the former goal is obviously desirable, and perhaps within reach, we thought it prudent not to tackle it before getting some understanding of the effect of coherence, which seems to be disasterous, e.g. for Runge-Kutta methods. We rather used some methods based on Taylor expansions that, besides having less serious coherence problems than R.K. methods, can be used to expand to any order.

We will try to explain the algorithms with a notation as close as possible to the computer programs even if this will occasionally lead to some small inconsistencies.

Problem I

$$(I) \begin{cases} D^2\psi + \beta D\psi + \dfrac{\alpha}{1 + \epsilon e^t}\psi = 0 \\ D\psi \sim \mu\psi \quad \text{as} \quad t \to -\infty. \end{cases}$$

We try to find an asymptotic expansion for ψ of the form $\psi = \sum \epsilon^n \psi_n(t)$, and show error bounds for the truncation. Expanding the fraction as a

geometric series we are led to

$$D^2\psi + \beta D\psi + \alpha\Psi = -\alpha \sum_{n=1}^{\infty} (-\epsilon)^n e^{nt}\psi$$

and we can recursively solve for ψ_n matching the corresponding powers of ϵ. It is easy to show by induction (except in some cases to be discussed later) that

$$\psi_n(t) = \eta_n e^{(u+n)t}, \qquad \eta_n \in \mathbb{R}.$$

Calling $P(x) = x^2 + \beta x + \alpha$, the characteristic polynomial of the R.H.S., we have

$$P(u+n)\eta_n = -\alpha \sum_{k=1}^{n} (-1)^k \eta_{n-k}.$$

Provided that $P(\mu) = 0$, this allows us to choose $\eta_0 = 1$ and then, all the other η's are determined provided $P(u+n) \neq 0$ for any n, which will be the case in our problem.

Even if this hierarchy is sufficient to compute as many terms as we wish we can transform it in such a way that computing the n^{th} term requires 1 operation and not n. This improves the execution time but, more importantly for us, the accuracy of the estimates.

In fact,

$$\begin{aligned} P(u+n)\eta_n &= -\alpha\left[-\eta_{n-1} + \sum_{k=2}^{n} (-1)^k \eta_{n-k} \right] \\ &= -\alpha\left[-\eta_{n-1} - \sum_{k=1}^{n-1} (-1)^k \eta_{n-1-k} \right] \\ &= [\alpha - P(u+n-1)]\eta_{n-1}. \end{aligned}$$

This recursion relation has the property that from a certain term on (which is 4 for our case) the η_n's decrease and hence, the series of the function and the derivatives will be uniformly convergent in all intervals of t of the form $\epsilon e^t \leqslant 1 - \delta$, $\delta > 0$. Hence, the expansion will be a solution of the O.D.E. in this range and have, clearly, the right asymptotic behavior.

ψ, ψ' at any point can be recovered by $\psi(t) = \left[\sum_{n=0}^{\infty} \eta_n (\epsilon e^t)^n \right] e^{\mu t}$

$$\psi'(t) = \sum_{n=0}^{\infty} \eta_n (\mu + n)(\epsilon e^t)^n e^{\mu t}.$$

Since for the discriminants, only the ratio matters, it suffices to compute the terms in brackets, which we evaluate using Horner's scheme. We sum a

finite number of terms and estimate the remainders using the decrease of $|\eta_n|$ and the sum of a geometric series

$$\left|\sum_{n=N+1}^{\infty}\eta_n(\epsilon e^t)^n\right| \leqslant |\eta_N|\sum_{n=N+1}^{\infty}(\epsilon e^t)^n = |\eta_n|(\epsilon e^t)^{N+1}\frac{1}{1-\epsilon e^t}$$

$$\left|\sum_{n=N+1}^{\infty}(\mu+n)\eta_n(\epsilon e^t)^n\right| \leqslant |\eta_N|\sum_{n=N+1}^{\infty}(\mu+n)(\epsilon e^t)^n$$

$$= |\eta_N|\left[\frac{(\mu+N+1)}{1-\epsilon e^t}+\frac{\epsilon e^t}{(1-\epsilon e^t)^2}\right](\epsilon e^t)^{N+1}.$$

Notice that this computation has the good feature that provided that we deal with the bounds on the input quantities using interval arithmetic we obtain bounds for the results.

This is implemented with the subroutines charpol, findetal, findf, findfl. The only nontrivial programming consideration is that we passed eta by address rather than by value. Even if this is slightly more dangerous, it saves the copying of the array η which takes quite an amount of time.

Remarks. In some cases, this algorithm could be used to propagate the solutions because we could very well produce two independent solutions, and we could find a linear combination to match the initial data. This is not enough to carry the computation for all cases of interest, but it can be used as a test. We have a program to perform this check in asymptest.

Now we turn to the problem II, solving the initial value problem.

It will be more convenient to use $\hat{\psi} = e^{2t}\psi$ which satisfies

$$\hat{\psi}'' + \left[\frac{\alpha}{1+\epsilon e^t} - 4\right]\hat{\psi} = 0.$$

(We will, for this discussion, forget the $\hat{}$ from now on.)

We first remark that, for this equation, it suffices to solve the initial value problem at $t_0 = 0$. Starting at another point is the same as substituting ϵe^{t_0} for ϵ.

The algorithm we are going to use will be based on expansions in powers of $(e^t - 1)$. We will try to find $\{\psi_n\}$ so that $\psi(t) = \sum_{n=0}^{\infty}\psi_n(e^t-1)^n$ solves the initial value problem.

If we write

$$\left[\frac{\alpha}{1+\epsilon e^t} - 4\right] = \sum_{n=0}^{\infty}\beta_n(e^t-1)^n$$

we have that the original equation is equivalent to

$$n^2\psi_n + (n+1)(2n+1)\psi_{n+1} + (n+1)(n+2)\psi_{n+2} + \sum_{k=0}^{\infty} \beta_k \psi_{n-k} = 0.$$

The β_n's can be readily computed using geometric series, and they are:

$$\begin{aligned} \beta_n &= \hat{\beta}_n \quad \text{if} \quad n \neq 0 \\ &= \hat{\beta}_0 - 4 \quad \text{if} \quad n = 0 \end{aligned}$$

where

$$\hat{\beta}_n \equiv (-1)^n \left(\frac{\epsilon}{1+\epsilon}\right)^n \frac{\sigma}{1+\epsilon}.$$

Besides, matching the initial conditions, we have $\psi_0 = \psi(0)$; $\psi_1 = \psi'(0)$. [Remember that our present $\psi(t)$ is what we called $e^{2t}\psi(t)$ in the original equation.]

So we can find recursively all ψ_n's. It will, however, be more convenient to proceed as we did before and use the relation between the β_n's to obtain a simpler recursion relation between the ψ_n's.

Writing explicitly the first term in the summation sign, pulling out the factors and substituting the previous equation of the hierarchy, we obtain

$$\begin{aligned} \psi_{n+2}(n+2)(n+1) = &-(n+1)(2n+1)\psi_{n+1} - (n^2 + \beta_0)\psi_n \\ &+ \left(\frac{-\epsilon}{1+\epsilon}\right)[n(n+1)\psi_{n+1} + n(2n-1)\psi_n + ((n-1)^2 - 4)\psi_{n-1}] \end{aligned}$$

which has to be supplemented by the initial conditions

$$\psi_0 = \psi(0);$$

$$\psi_1 = \psi'(0);$$

$$\psi_2 = \frac{1}{2}[-\beta_0\psi_0 - \psi_1];$$

the third one being, of course, the second equation of the original hierarchy. It becomes necessary because one equation of the new hierarchy is a combination of two consecutive ones of the previous, hence of higher order.

Again, when we recursively compute in interval arithmetic, we obtain bounds for the ψ_n's uniform in the input parameters ranging in the given intervals.

Now we turn to the task of bounding the error incurred by truncating the hierarchy to a finite order.

We rewrite the equation in integral form as

$$\psi = \psi_0 + I(\psi_1 + IH\psi) \equiv K\psi$$

where I is the operator $\psi \to \int_0^x \psi(s)\,ds$. H is multiplication by $[\alpha/(1 + \epsilon e^t) - 4]$. Those operators act on the spaces of functions $\{\psi = \sum \psi_n (e^t - 1)^n | \, \|\psi\| = \sum |\psi_n| r^n < \infty\}$ where r is an arbitrary parameter to be chosen later.

The idea of the proof will be to estimate, for our finite dimensional approximation ψ, $\|\psi - K\psi\|$ and show that, for sufficiently small r, K is a contraction. Using these estimates, we can bound $\|\psi_{\text{true}} - \psi\| \leqslant \|\psi - K\psi\|/(1 - Lip(K))$ and a fortiori the error for the ψ and ψ' evaluated in a place where $|e^t - 1| < r$.

Proposition 1.

$$\|I\| \leqslant \frac{r}{1 - r}.$$

PROOF. $(\psi')_n = n\psi_n + (n + 1)\psi_{n+1}$. Hence $(n + 1)(I\psi)_{n+1} = \psi_n - n(I\psi)_n$, so $(n + 1)(I\psi)_{n+1} = \psi_n - n(I\psi)_n = \psi_n - \psi_{n-1} + \psi_{n-2} + \cdots \pm \psi_0 \cdot I\psi_0 = 0$.

Therefore

$$\sum_{n=0}^{\infty} |(I\psi)_n| r^n \leqslant \sum_{n=1}^{\infty} \frac{r^n}{n} \sum_{k=0}^{n} |\psi_k|.$$

The derivative of the R.H.S. with respect to r is $\sum_{n=0}^{\infty} r^n \sum_{k=0}^{n+1} |\psi_k|$, but this is just $\left(\sum |\psi_k| r^k\right)[1/(1 - r)]$. Integrating and noting that at $r = 0$ this R.H.S. takes the value 0, we obtain

$$\|I\| \leqslant \int_0^r \frac{ds}{1 - s} \leqslant \int_0^r \frac{ds}{1 - \mathrm{r}} = \frac{r}{1 - r}.$$

We also observe that if $\psi(t) = O(t^N)$, $I\psi = O(t^{N+1})$ near $t = 0$.

Proposition 2.

$$\|H\| \leqslant \left|\frac{\alpha}{1 + \epsilon}\right| \left(1 - \left|\frac{\epsilon r}{1 + \epsilon}\right|\right)^{-1} + 4.$$

PROOF. It is well known that these spaces of analytic functions will have the Banach algebra property under multiplication. Therefore, the norm of a multiplication operator is less than the norm of the function. The R.H.S. of the formula above is a bound of the function H in view of the explicit expansion computed.

Clearly, the Lipschitz constant of K can be bounded by $\|I\|^2 \|H\|$.

The bound for $\|\psi - K\psi\|$ is a little bit more subtle. We should remember that the intervals that we have produced for the ψ_n are only bounds for the values of ψ_n, but they are not all the information on ψ_n we are entitled to use. Particularly, we can use that the ψ_n satisfy the previous recursion relation exactly and, therefore, we can conclude that many of the terms to estimate are indentically zero.

In particular we have that, since the equation is satisfied to order N, $K\psi - \psi = K\psi^{[>N]} - \psi^{[>N]} = K\psi^{[>N]}$ because $\psi^{[>N]} = 0$.

If $N \geqslant 2$ —as we will always assume— we have

$$\|(I^2 H\psi)^{[>N]}\| \leqslant \|I\|^2 \|(H\psi)^{[>N-2]}\|$$

$$\|(H\psi)^{[>N-2]}\| \leqslant \sum_{n=N-1}^{\infty} r^n \sum_{k=0}^{n} |\beta_{n-k}|\,|\psi_k|$$

$$\leqslant \sum_{n=N-1}^{\infty} r^n \left(\sum_{k=0}^{n} |\hat{\beta}_{n-k}|\,|\psi_k| + 4|\psi_n| \right)$$

$$= \left|\frac{\alpha}{1+\epsilon}\right| \sum_{n=N-1}^{\infty} r^n \sum_{k=0}^{n} \left|\frac{\epsilon}{1+\epsilon}\right|^{n-k} |\psi_k| + 4|\psi_{N-1}| r^{N-1}$$

$$+ 4|\psi_n| r^N$$

Using that ψ_n is zero for $n > N$, we have that the first term can be rewritten as follows, where $\gamma = |\epsilon/(1+\epsilon)|$:

$$\left|\frac{\alpha}{1+\epsilon}\right| r^{N-1} \sum_{k=0}^{N-1} \gamma^{N-1-k} |\psi_k| + \left|\frac{\alpha}{1+\epsilon}\right| \sum_{n=N}^{\infty} r^n \gamma^{n-N} \sum_{k=0}^{N} \gamma^{N-k} |\psi_k|.$$

The last term can be bounded by

$$\left|\frac{\alpha}{1+\epsilon}\right| \frac{r^N}{1-r\gamma} \left(\sum_{n=0}^{N} \gamma^{n-k} |\psi_k| \right) = \left|\frac{\alpha}{1+\epsilon}\right| \frac{r^N}{1-r\gamma} (S\gamma + |\psi_N|)$$

where

$$S = \sum_{k=0}^{N-1} \gamma^{N-1-k} |\psi_k|.$$

Putting everything together, we obtain

$$\|K\psi - \psi\| \leqslant r^{N-1} \left[\left|\frac{\alpha}{1+\epsilon}\right| \left\{ S\left(1 + \frac{\gamma r}{1-\gamma r}\right) + \frac{|\psi_N| r}{1-\gamma r} \right\} + 4\{|\psi_N| + |\psi_{N-1}|\} \right].$$

To bound the truncation error we can use not only the estimate on norm of $\psi_{\text{true}} - \psi$ but also the fact that this function is of high order. We have

$$|\psi_{\text{true}}(\delta) - \psi(\delta)| \leqslant \left(\frac{\delta}{r}\right)^{N+1} \|\psi_{\text{true}} - \psi\|$$

$$|\psi'_{\text{true}}(\delta) - \psi(\delta)| \leqslant \left[\sup_{n \geq N+1} n\left(\frac{\delta}{r}\right)^{n} + \sup_{n \geq N+1} (n+1)\frac{\delta^n}{r^{n+1}}\right] \|\psi_{\text{true}} - \psi\|$$

as can be seen by writing the sums explicitly, and bounding in the crudest way. We observe that if δ/r is sufficiently small, the sup is reached in the first term and the bracket in front of the derivative becomes

$$(2N+3)\left(\frac{\delta}{r}\right)^{N+1}\left(1+\frac{1}{r}\right).$$

Notice that $\delta/r = 1/2$ is sufficiently small for the purpose of having the previous property independently of N.

Notice that the previous bound runs into problems when $r = 0$ (numerically, it would be disastrous when r is very small). So the obvious choice of r as 2δ would lead to problems when δ is very small. Hence, arbitrarily, we have decided that when $\delta < 10^{-6}$, then we take $r = 10^{-4}$ (or, more precisely, the numbers the compiler interprets when given 10^{-6} and 10^{-4}; we do not care about computing them carefully since they are just arbitrary choices and the only thing we assume is that one is more than double the other).

Remark. The way of grouping terms in the previous computation is quite important. For certain ranges of the parameter, γ can be extremely small and pulling γ^N out of the sum may lead to disastrous results. This at first sight could be quite puzzling since the estimates become ridiculous at some points where γ comes close to zero.

Unfortunately, these things seem to be quite hard to guess without actually writing and running the program.

The only place where such difficulty would show up would be requesting steps acting on two identical sharp intervals as initial and final time, and clearly those could be avoided in the program. However, we have decided to have a subroutine that works even in the absurd cases. This allows us to use the same general propagation subroutine without worrying about the boundary conditions in do loops which lead to these special cases.

In the subroutine step, we take as inputs, the initial and the final time, and obtain an interval containing all the increments. We compute $(e^{\Delta t} - 1)$ and evaluate a finite truncation of the propagation.

There is another boolean variable «inconclusive» that gets set if any of the geometric series estimates we were performing has ratio bigger than 1. This

will inform the main program that something wrong has happened: the data should be discarded, the increment halved and start all over (this turns out to be more convenient that just stopping the program).

There are many advantages to organizing step in the way we did. Being as conservative as possible in the lowest level saves considering which way the uniformity of the estimates goes in the higher levels.

Clearly, we can divide the initial value problem for large intervals in a sequence of smaller steps. Notice that, with our organization, it suffices that the final time of one step contains the initial time of the next and is not necessary that they are equal.

Nevertheless, it cannot be denied that basing a step in the initial interval and the increment would have had something more natural to it, since the increment is really what is used. However, the final interval that we can guarantee should be an interval contained (not containing as before!) in the sum. This would require getting out of the philosophy of interval arithmetic, implementing new operations, and we thought it unappealing and confusing and stuck to the safe —albeit wasteful— procedure. This, we admit, we could do because the problem was not too delicate. It could very well be that more demanding problems require improvements in this direction, extending interval arithmetic with some inverse operations.

Out of this, it is quite trivial how to propagate the solutions from zero and infinity and compute if, for some given range of parameters, there is a match. This is what is done by the boolean function «excluded». It is natural to give it three parameters, t_1 and t_2 and a third one we called offset, which, when it takes the value zero, makes us recover the first problem and that by varying in an interval produces the changes in ϵ required by the other problems.

Given those parameters, this function computes the numerator of the discriminant, taking good care that each time step is involved, it is tested whether it is conclusive or not. The only non-obvious point is that we propagate taking sharp intervals as intermediate steps, so that we are only forced to have steps with fat intervals as ends at t_1 and t_2 themselves. For those we found it advantageous to compute in three different ways from a sharp interval in the middle and from sharp intervals at each end and take the intersection.

Switching differential equations is a little bit cumbersome. Notice that here it is not sufficient that the initial time of the next step is contained in the final time of the previous one; the reason is that t_2 has two roles: one as final time of the propagation and also as the place where the differential equation changes. Even if our bounds would allow us to restrict the final time of propagation, the say nothing about the effect of switching O.D.E. at a different time.

The solution is to propagate the new O.D.E. from the full t_2 to another interval, a little bit ahead. After that the propagation is very similar to the one used before.

At the end, «excluded» prints the numerator of the discriminant and the interval in which it was invoked. Even if the input and output —which are not quite right— are invoked to do that, this does not affect the conclusions because the numbers written are only going to be used as heuristic aids.

The only thing left now is to explore in a systematic way all the range to be excluded.

The most obvious way would be to just make a double loop exploring in turn small squares adding up to the whole range to be explored. This, however, would be quite impractical since the allowable size for the estimates to be informative depends very much on the place and this cannot be guessed before doing the computation.

A clean way out is to introduce another function: «superexc» that, given a certain square in parameter space, will compute the discriminant and if it cannot exclude a match, will perform a subdivision of the original square in four squares and if any of those cannot be excluded will again subdivide and so on. It will only report false when one of the squares to be considered is very small (we set it at 10^{-4}).

The way to implement this in Pascal is through a recursive function that, if it does not succeed with excluded, and the intervals are not too small, subdivides and invokes itself on each of the subdivisions.

This simple trick turns out to be extraordinarily effective and is quite a useful structure for exclusion problems.

Remark. For the aficionadoes of fine points in compilers it may be amusing to notice that the efficiency with which superexc would abandon when confronted with a non excludable interval depends quite dramatically on the way that the compiler implements «and».

In Modular-2, for example ([W] p. 27) «p AND q» is taken to mean «IF p THEN q ELSE FALSE». This has the advantage that if p turns out to be false, q does not need to be evaluated. Similar conventions are specified in C ([K.R.] p.190) and, several Pascal compilers adhere to it even if nothing is said in the specification of the language [JW].

Other compilers, however, attempt to evaluate both p and q before evaluating the «and».

If the first alternative is in effect, superexc would abandon the first time it needs to consider a very small interval whereas if the second alternative is in effect superexc will abandon the last time it is so required.

For our case, with two variables and too small meaning 100 times smaller than typical, this would mean roughly $(100)^2$ times slower. If all intervals had to be examined but even if there were matches only along a line, this would mean 100 times slower.

On top of that, we just put another loop than invokes superexc on a sequence of squares of moderate size and covering the interval to be excluded. After each one of them is excluded, a report is put into a file and, at the end, it would be quite easy to check it is there using an editor. This turns out to be safer than reporting if something bad happens, because power failures could happen that would prevent messages from being issued, and this would be interpreted wrongly as success. These reports of success can be extracted from the other printout of the program very easily using a filter («grep» in Unix). Similarly, we could extract the report about status.

On other systems, it could be advantageous to write the heuristic reports and the definitive ones to different files.

Notice that the loop proceeds dividing in intervals by multiplying an integer by a conventional number which, strictly speaking, we do not know. (We only know it is the number the compiler translates 0.1 .) But it does not matter because the upper point of one interval is the lower point of the next. (We refrained from assuming that either the translation algorithm or the multiplication is monotonic.)

Remark. If no matches occur, the procedure we have explained will prove that no matches occur. If there were a match, however, the procedure would keep on dividing till the subdivisions reach the cut-off in superrexc, which can be quite a long time and even in that case we could not prove that indeed there was a zero. However, it is possible —and indeed we did it in a previous version— to run the exploration from two opposite corners towards the center and keep a global variable of the signs found so that the existence of a match could also be established and even located very accurately. Since it was not necessary in our case and it gives some room for error, we proceeded in the most straightforward way.

Besides the loop cycling through squares, the main program asks for several parameters that, even if logically irrelevant, will affect the efficiency of the run, like the number of terms to keep or the number of steps to take. Since trying to run through the whole loop at once would be quite impractical, we just run a few integers at a time.

We have written four main programs, «veuler», «suffices», «lastbound», «nomatchØ» which correspond to the statements claimed in Lemma 4.

Each one of them has its own choice of parameters of the run, its own superexc and mainloop, but they share most of the other subroutines and we keep them as «include» files.

From the programming point of view, it is instructive to point out that the use of the same excluded for the three first leads to «lastbound» requesting a propagation form the interval vØ to itself; and, hence, it becomes convenient to have a «step» that works correctly even in this case.

The last bound can be considerably simplified. Excluding matches $t_1 = t_2 = 0$, $2.25 \leqslant \epsilon \leqslant 4.5$ is equivalent to $G(0) = 0$ and ϵ ranging over this interval means that $\epsilon_g = \frac{1}{5}[\epsilon - 2 - e^{-t_1} - 3e^{-t_2}]$ ranges over $(-3/4, -3/10)$ and this is what «matchØ» accomplishes.

The first two programs run much slower than the last two but, in a VAX 11/750, with lowest priority, they both run in a few days. The last two can be completed in a few minutes.

«veuler» run excluding matches for $-2.5 \leqslant t_1 \leqslant t_2 \leqslant 0.05$.

«suffices» has built in to take $T = 2.0$ and cycle from an interval in offset $5/4 + 0.05$ to -0.05.

«lastbound» also has built in to take $T = 2.0$ and we run it from offset $5/4 - 0.05$ to $5/4 + 0.05*26$.

«no matchØ» was run from offset $(-8)*0.1$ to $(-2)*0.1$.

Notice that both «suffices» and «lastbound» have a T generated in an identical way, so it is an identical number in both programs.

The other numbers involved are generously overlapping so that they cover by far the possible inaccuracies of the input and output subroutines.

Even after the algorithm described before is coded in Pascal, there are several steps towards its execution: the program has to be translated into something the machine understands —compiled— and the compiled versions have to be shifted from memory to disk and scheduled to share the resources with other processes. One could worry that we do not have much control over these steps.

It can be argued that nowadays developing compilers is a well understood technique and, when not worried about optimizing resources, (e.g., in personal computers or with optimizers) the results are very reliable.

Besides, we did some checking on the parts of the compiler we used and refrained from using delicate parts like optimizers or separate compilation. We run the program in two machine equipped with Berkeley 4.1 BSD and with ULTRIXVI.0 (4.2 BSD), which are developed independently and we could compare parts of the output. This excludes also the possibility of some random malfunction or strange interference with other processes.

In spite of all this checking, however, the weakest part is the human effort of programming. (That is why we decided to publish the programs so that they can be checked by anybody wanting to do so.) Nevertheless, we feel more or less confident that there are at most minor errors; certainly there are not terribly serious errors because, besides checking the program we wrote heuristic programs to compute discriminants based on different algorithms, in Pascal, Fortran and Basic. Even if the effort was independent and conducted in different computers the results agreed. The existence of niggling errors in the estimates cannot be so easily excluded —and, of course, it is logically very serious and we feel strongly about it. However, the programming was very

careful and all the algorithms worked correctly at the first try —only once we found a faulty reasoning. In any case, were any error to appear, the estimates are so conservative that it would certainly be absorbable at the expense, at most, of more computing time; at the moment, running the whole verification is not a problem at all.

Cetainly we want to encourage our colleagues to check the programs and we can make them available, including some tests, through USNET or BITNET, or we can send an I.B.M. diskette (double-sided, double-density). It goes without saying that any error reported to us will be greatly appreciated.

We believe that this amount of reliability is quite comparable to any mathemtical proof involving complicated computation. There is always the possibility of error; minor errors will probably never be entirely removed. Nevertheless, after careful checking one has very strong reasons for confidence. Except for philosophical objections we will not address in public, Lemma 4 should be considered as rigorously established.

Acknowledgments

We thank A. Casado for providing access to ADRVAX where a confirmation run of our results was performed. He and D. Rana provided much information about UNIX. The development, and running of our programs was performed in a computer supported by N.S.F. Grant MCS82-01604.

References

[Mo] Moore, R. E. Methods and applications of interval analysis, *S.I.A.M.*, Philadelphia (1979).

[K-M] Kaucher, E. W., and Miranker, W. L. Self-validating numerics for function space problems, *Academic Press*, New York (1984).

[L] Lanford II, O. E. *Bul.* AMS 6, 3 (1982), 427-434.

[E-K-W] Eckmann, J. P., Koch, H., Wittwer, P. A computer assisted proof of universality, *Memoirs A.M.S.*

[E-W] Eckmann, J. P., Wittwer, P. To be published by *Springer-Verlag Lecture Notes in Physics*.

[M-P] Mckay, R. S., Percival, I. Converse K.A.M. theory and pactice, to appear in *Comme. Math. Phys.*

[L-L] Lanford III, O. E., de la Llave, R. to appear.

[Me] Mestel, B.

[T] Tannenbaum, A. S. Structured computer orgnaization, 2nd ed., *Prentice Hall*, Englewood Cliffs (1984).

[DEC] Digital Equipment Corporation VAX architecture handbook, *Digital Equipment Corporation* (1981).

[J-W] Jensen & Wirth Pascal user's manual and report, 3rd ed., *Springer-Verlag*, New York (1984.

[W] Niklaus Wirth, Programming in Modula-2, Second edition, *Springer-Verlag*, New York (1982).

[K-R] Kernigham, B. W., and Ritchie, D. M. The C programming language, *Prentice Hall*, 1978.

[C] Conlon, J. *Comm. Math. Phys.*, **94** (1984), 439.

[DL 1] Dyson, F. and Lenard, A. *J. Math. Phys.*, **8** (1967), 423; **9** (1968), 698.

[B] Baxter, Ill. *J. Math.*, **24** (1980), 645-652.

[DL2] Daubechies I. and Lieb, E. *Comm. Math. Phys.*, **90** (1983), 511.

[F] Federbush, P. *J. Math Phys.*, **16** (1975), 706.

[H] Herbst, I. *Comm. Math. Phys.*, **53** (1977), 285; **55** (1977), 316.

[L] Lieb, E. *Rev. Mod. Phys.*, 53 no. 4 part 1, 1981.

[LT] Lieb, E. and Thirring, W. *Phys. Rev. Lett.* **35** (1975), 687.

[LT_2] Lieb, E. and Thirring, W. Gravitational collapse in Quantum mechanics with relativistic kinetic energy, *Annals of Phys.* 155 no. **2** (1984), 494.

C. Fefferman and R. de la Llave
Princeton University
Princeton, NJ 08544
U.S.A.

veuler

```
program veuler ( input , output );
const
#include 'const.i'
type
#include 'types.i'

var
#include 'globalround.i'
#include 'global.i'
        orderused :integer;
        t1,t2 :interval;
        etag :list;
        ng :integer;
        alphaf,alphah :interval;
        vrootg :interval;
        sucess :boolean;
        stepnum1,stepnum2 :integer;
        i,j :integer;
        imin,imax :integer;
        demult :integer;
#include 'complex.i'
#include 'cexpo.i'
#include 'initialize.i'
#include 'asymp.i'
#include 'step.i'
#include 'excluded.i'
```

superexc

```
function superexc( t1 , t2 :interval):boolean;
label
        1;
var
        t1center,t2center :double;
        t1plus,t1minus,t2plus,t2minus :interval;
begin
        if excluded( t1 , t2 , v0) then
        begin
                superexc := true;
                goto 1;
        end
        else
        begin
                if (meas(t1) < 1.0e-04) and( meas( t2) < 1.0e-04 )then
                begin
                        superexc := false;
                        goto 1;
                end;
```

```
                t1center := center(t1);
                t1plus.lower := left( t1center );
                t1plus.upper := right( t1.upper );
                t1minus.upper:= right( t1center );
                t1minus.lower:= left( t1.lower );
                t2center := center( t2 );
                t2plus.lower := left( t2center );
                t2plus.upper := right( t2.upper );
                t2minus.upper:= right( t2center );
                t2minus.lower:= left( t2.lower );
                superexc :=
                        superexc( t1plus , t2plus  )and
                        superexc( t1plus , t2minus  )and
                        superexc( t1minus , t2plus  )and
                        superexc( t1minus , t2minus  );
        end;
1:
end(* superexc *);
begin
        initialize;
        writeln( 'key in the number of terms to keep for G' );
        readln (ng );
        writeln( ' ng = ', ng );
        writeln( 'key in the number of terms to be used');
        readln( orderused );
        writeln( ' orderused = ',orderused );
        writeln ( 'key in imin ');
        readln( imin );
        writeln( 'imin = ',imin );
        writeln ( ' key in imax ');
        readln ( imax );
        writeln ( 'imax = ', imax );
        writeln( 'key in the demultiplication factor ');
        readln ( demult );
        writeln ( 'demult =  ', demult );
        vroot( 2 , v2over5 , vrootg );
        subv( vrootg , v2 , vrootg );
        findeta( vrootg , v18over5 , v4 , etag , ng );
        alphaf := v6;
        alphah := v9over2;
        for i := imin to imax do
        begin
                t1.lower := -i*0.1;
                t1.upper := -(i-1)*0.1;
                t1.lower := left( t1.lower );
                t1.upper := right( t1.upper );
```

```
                for j := 1 to i do
                begin
                        t2.lower := -j*0.1;
                        t2.upper := -(j-1)*0.1;
                        t2.lower := left( t2.lower );
                        t2.upper := right( t2.upper );
                        stepnum2 := demult*j;
                        stepnum1 := demult*(i-j+1);
                        sucess := superexc ( t1 , t2 );
                        if sucess then
                                writeln( i , j ,'superexcluded')
                        else
                                writeln( i , j ,'NOT SUPEREXCLUDED' );
                end;
        end;
        writeln;
        writeln( ' status = ', status);
        writeln;
end.
```

suffices

```
program suffices ( input , output );
const
#include 'const.i'
type
#include 'types.i'
var
# include 'global.i'
# include 'globalround.i'
        orderused :integer;
        bigt,t2 :interval;
        alphaf,alphah :interval;
        etag :list;
        ng :integer;
        vrootg :interval;
        sucess :boolean;
        stepnum1,stepnum2 :integer;
        i,j :integer;
        imin,imax :integer;
        demult :integer;
        offset :interval;
#include 'complex.i'
#include 'cexpo.i'
#include 'initialize.i'
#include 'asymp.i'
```

superexc

```
#include 'step.i'
#include 'excluded.i'
function superexc(  t2 :interval; offset :interval):boolean;
label
        1:
var
        offsetcenter,t2center :double;
        offsetplus,offsetminus,t2plus,t2minus :interval;
begin
        if excluded( bigt . t2 , offset ) then
        begin
                superexc := true;
                goto 1;
        end
        else
        begin
                if (meas(offset) < 1.0e-04) and( meas( t2) < 1.0e-04 )then
                begin
                        superexc := false;
                        goto 1;
                end;
                offsetcenter := center(offset);
                offsetplus.lower := left( offsetcenter );
                offsetplus.upper := right( offset.upper );
                offsetminus.upper:= right( offsetcenter );
                offsetminus.lower:= left( offset.lower );
                t2center := center( t2 );
                t2plus.lower := left( t2center );
                t2plus.upper := right( t2.upper );
                t2minus.upper:= right( t2center );
                t2minus.lower:= left( t2.lower );
                superexc :=
                        superexc(  t2plus , offsetplus )and
                        superexc(  t2minus , offsetplus )and
                        superexc(  t2plus , offsetminus )and
                        superexc(  t2minus , offsetminus );
        end;
1:
end(* superexc *);
begin
        initialize;
        writeln( 'key in the number of terms to keep for G' );
        readln (ng );
        writeln( ' ng = ', ng );
        writeln( 'key in the number of terms to be used');
```

```
        readln( orderused );
        writeln( ' orderused = ',orderused );
        writeln ( 'key in imin ');
        readln( imin );
        writeln( 'imin = ',imin );
        writeln ( ' key in imax ');
        readln ( imax );
        writeln ( 'imax = ', imax );
        writeln( 'key in the demultiplication factor ');
        readln ( demult );
        writeln ( 'demult =  ', demult );
        vroot( 2 , v2over5 , vrootg );
        subv( vrootg , v2 , vrootg );
        findeta( vrootg , v18over5 , v4 , etag , ng );
        alphaf := v6;
        alphah := v9over2;
        bigt := vm2;
        for i := imin to imax do
        begin
                t2.upper := -(i-1)*0.05;
                t2.lower := -i*0.05;
                t2.lower := left( t2.lower );
                t2.upper := right( t2.upper );
                stepnum2 := trunc( -t2.lower/0.05)+1;
                stepnum1 := trunc( ( -bigt.lower + t2.upper)/0.05) +1;
                stepnum1 := stepnum1 * demult;
                stepnum2 := stepnum2 * demult;
                for j := 0 to 26 do
                begin
                        offset.lower := ( j-1 )*0.05;
                        offset.upper := (j )*0.05;
                        sucess := superexc ( t2 , offset );
                        if sucess then
                                writeln( i , j ,'superexcluded')
                        else
                                writeln( i , j ,'NOT SUPEREXCLUDED' );
                end;


        end:
        writeln;
        writeln( ' status = ', status);
        writeln;
end.
```

nomatch0

```
program nomatch0( input , output );
const
#include 'const.i'
type
#include 'types.i'
var
        etag :list;
        ng :integer;
        vrootg :interval;
        sucess :boolean;
        i :integer;
        imin,imax :integer;
        epsg :interval;
#include 'global.i'
#include 'globalround.i'
#include 'complex.i'
#include 'cexpo.i'
#include 'initialize.i'
#include 'asymp.i'
function superexc( eps :interval ):boolean;
label
        1;
var
        epscenter:double;
        epsplus,epsminus:interval;
        gat0 :interval;
        excluded :boolean;
begin
        findf0( etag , ng , eps , gat0);
        excluded :=  not( contained( v0 , gat0 ));
        write( 'eps = ');printv( eps );writeln;
        write( 'gat0 = ');printv( gat0 );writeln;
        writeln( 'excluded = ', excluded );
        writeln;writeln;
        if  excluded then
        begin
                superexc := true;
                goto 1;
        end
        else
        begin
                if (meas(eps) < 1.0e-04) then
                begin
                        superexc := false;
                        goto 1;
```

superexc

```
                end;
                epscenter := center(eps);
                epsplus.lower := left( epscenter );
                epsplus.upper := right( eps.upper );
                epsminus.upper:= right( epscenter );
                epsminus.lower:= left( eps.lower );
                superexc :=
                        superexc( epsplus  )and
                        superexc( epsminus );
        end;

1:
end(* superexc *);
begin
        initialize;
        writeln( 'key in the number of terms to keep for G' );
        readln (ng );
        writeln( ' ng = ', ng );
        writeln ( 'key in imin ');
        readln( imin );
        writeln( 'imin = ',imin );
        writeln ( ' key in imax ');
        readln ( imax );
        writeln ( 'imax = ', imax );
        vroot( 2 , v2over5 , vrootg );
        subv( vrootg , v2 , vrootg );
        findeta( vrootg , v18over5 , v4 , etag , ng );
        for i := imin to imax do
        begin
                epsg.upper := i*0.1;
                epsg.lower := (i-1)*0.1;
                sucess := superexc ( epsg );
                if sucess then
                begin
                        writeln( i , 'superexcluded')
                end
                else
                begin
                        writeln( i , 'NOT SUPEREXCLUDED' );
                        writeln;
                end;
        end;
        writeln( ' status = ', status);
        writeln;
end.
```

lastbound

```
program lastbound ( input , output );
const
#include 'const.i'
type
#include 'types.i'
var
#include 'globalround.i'
# include 'global.i'
        orderused :integer;
        bigt :interval;
        offset :interval;
        alphaf,alphah :interval;
        etag :list;
        ng :integer;
        vrootg :interval;
        sucess :boolean;
        stepnum1,stepnum2 :integer;
        i :integer;
        imin,imax :integer;
        demult :integer;
#include 'complex.i'
#include 'cexpo.i'
#include 'initialize.i'
#include 'asymp.i'
#include 'step.i'
#include 'excluded.i'
function superexc(offset :interval ):boolean;
label
        1;
var
        offcenter:double;
        offplus,offminus:interval;
begin
        if excluded( bigt , v0 , offset ) then
        begin
                superexc := true;
                goto 1;
        end
        else
        begin
                if (meas(offset) < 1.0e-04) then
                begin
                        superexc := false;
                        goto 1;
                end;
```

superexc

```
                offcenter := center(offset);
                offplus.lower := left( offcenter );
                offplus.upper := right( offset.upper );
                offminus.upper:= right( offcenter );
                offminus.lower:= left( offset.lower );
                superexc :=
                        superexc( offplus  )and
                        superexc( offminus );

        end;
1:
end(*  superexc *);
begin
        initialize;
        writeln( 'key in the number of terms to keep for G' );
        readln (ng );
        writeln( ' ng = ', ng );
        writeln( 'key in the number of terms to be used');
        readln( orderused );
        writeln( ' orderused = ',orderused );
        writeln ( 'key in imin ');
        readln( imin );
        writeln( 'imin = ',imin );
        writeln ( ' key in imax ');
        readln ( imax );
        writeln ( 'imax = ', imax );
        writeln( 'key in the demultiplication factor ');
        readln ( demult );
        writeln ( 'demult =  ', demult );
        vroot( 2 , v2over5 , vrootg );
        subv( vrootg , v2 , vrootg );
        findeta( vrootg , v18over5 , v4 , etag , ng );
        alphaf := v6;
        alphah := v9over2;
        bigt := vm2;
        for i := imin to imax do
        begin
                offset.upper := i*0.1;
                offset.lower := (i-1)*0.1;
                stepnum1 := 20*demult;
                stepnum2 := 1;
                sucess := superexc ( offset );
                if sucess then
                begin
                        writeln( i , 'superexcluded')
```

```
                end
                else
                begin
                        writeln( i , 'NOT SUPEREXCLUDED' );
                        writeln;
                end;
        end;
        writeln( ' status = ', status);
        writeln;
end.
```

```
(*TTHIS IS ASYMP.I *)                                          charpol
procedure charpol( alpha , beta :interval; x :interval ; var poly:interval);
begin
        poly := x;
        addv( poly , beta , poly );
        mulv( poly , x , poly );
        addv( poly , alpha , poly );
end(*  charpol *);
procedure findeta ( u ,alpha,beta :interval ; var eta :list; n:integer);   findeta
var
        i :integer;
        vi,vim1,upi,upim1,poli,polim1,factor :interval;
begin
        eta[0] := v1;
        for i := 1 to n do
        begin
                cvtiv( i , vi);
                cvtiv( i-1 , vim1 );
                addv( vi , u , upi );
                addv( vim1 , u , upim1 );
                charpol( alpha , beta , upi , poli);
                charpol( alpha , beta , upim1 , polim1 );
                subv( alpha , polim1 , factor );
                divv( factor , poli , factor , status );
                mulv( eta[i-1] , factor , eta[i] );
        end;
end(*  findeta *);                                                  findf0
(*  Notice that findf0 and findf1 assume that n is big enough so that they could
be incorrect if the sufficiency of n  is not checked before calling them *)
procedure findf0(var  eta :list; n :integer; exptepsilon :interval; var f0 :interval);
var
        i :integer;
        berror,berror1,berror2,berror3 :bound;
        verror :interval:
begin

        absv( exptepsilon , berror1);
        if cmpb( berror1 , b1 ) <> -1 then
        begin
                writeln( 'expanding in an unfavorble range' );
                halt;
        end;
        f0 := eta[n];
        for i := n-1 downto 0 do
```

```
begin
        mulv( f0 , exptepsilon , f0 );
        addv( f0 , eta[i] , f0 );
end;
absv( exptepsilon , berror1 );
powerb( n+1 , berror1 , berror1 );
absv( eta[n] , berror2 );
mulb( berror1 , berror2 , berror);
```

```
        subv( v1 , exptepsilon , verror );
        divv( v1 , verror , verror , status );
        absv( verror , berror3);
        mulb( berror , berror3 , berror );
        enlargev( f0 , berror , f0);
end(* findf0 *);
procedure findf1 ( u:interval;var  eta :list; n :integer; exptepsilon :interval;
  var f1 :interval);                                                    findf1

var
        i :integer;
        sum,vwork :interval;
        berror,berror1,berror2,berror3,berror4,berror5 :bound;
        verror,vnp1 :interval;
begin
        absv( exptepsilon , berror1 );
        if cmpb( berror1 , b1 ) <> -1 then
        begin
                writeln( 'expanding in a bad range' );
                halt;
        end;
        cvtiv( n , vwork);
        addv( vwork , u , vwork );
        mulv( eta[n] , vwork , sum );
        for i:= n-1 downto 0 do
        begin
                mulv( sum , exptepsilon , sum );
                cvtiv( i, vwork );
                addv( vwork , u , vwork );
                mulv( vwork , eta[i] , vwork );
                addv( sum ,vwork , sum );
        end;
        absv( exptepsilon , berror1 );
        powerb( n+1 , berror1 , berror2 );
        subv( v1 ,  exptepsilon , verror);
        divv( v1 , verror , verror , status );
        absv( verror , berror3 );
        cvtiv( n+1 , vnp1);
        addv( u , vnp1 , verror);
        absv( verror , berror4);
        absv( eta[n] , berror5 );
        mulb( berror1 , berror3 , berror );
        addb( berror4 , berror ,berror );
        mulb( berror , berror3 , berror );
        mulb( berror2 , berror , berror );
```

```
        mulb( berror5 , berror , berror );
        enlargev( sum , berror , f1 );
end(* findf1 *);
```

```
(   THIS IS CEXPO.I *)
procedure expvv( v :interval; var expo:interval);                    expvv
const
        degree = 12;
var
        i :integer;
        vi :interval;
        dwork :double;
begin
        if not( contained( v , v32) ) then
        begin
                writeln( 'evaluating the exponential out of range');
                halt;
        end;
        if not (  contained( v , width )) then
        begin
                divv( v , v2 , v , status );
                expvv( v , expo );
                mulv( expo , expo , expo );
        end
        else
        begin
                expo := v1;
                if ( v.lower < d0 ) then
                begin
                        cvtid( degree+1 , dwork );
                        dwork := v.lower/dwork;
                        dwork := left( dwork );
                        expo.lower := expo.lower + dwork;
                        expo.lower := left( expo.lower );
                end;
                if ( v.upper > d0 ) then
                begin
                        cvtid( degree+1 , dwork );
                        dwork := v.upper / dwork;
                        dwork := right( dwork );
                        dwork := d1 - dwork;
                        dwork := left( dwork );
```

```
                dwork := d1/dwork;
                expo.upper := right( dwork );
            end;
            for i := degree downto 1 do
            begin
                mulv( expo , v , expo );
                cvtiv( i , vi );
                divv( expo , vi , expo , status );
                addv( expo , v1 , expo );
            end;
        end;
end(*  expvv *);
procedure cosvv( v :interval; var vcos :interval);forward;          COSVV
(*  *)
procedure sinvv( v :interval; var vsinus :interval );               SINVV
const
        degree = 7;
var
        vwork1,vwork2,factori,vsquare :interval;
        i :integer;
        dwork :double;
begin
        if not( contained(v , v32) ) then
        begin
                writeln( 'evaluating vsin out of range');
                halt;
        end;
        if not ( contained( v , width ) ) then
        begin
                divv( v , v2 , v , status );
                cosvv( v , vwork1 );
                sinvv( v , vwork2 );
                mulv( vwork1 , vwork2 , vsinus );
                mulv( vsinus , v2 , vsinus );
        end
        else
        begin
                mulv( v , v , vsquare );
                vsinus := v1;
                cvtid( (2*degree + 2 )*(2*degree + 3) , dwork );
                dwork := vsquare.upper/dwork;
                dwork := right( dwork );
                vsinus.lower := d1 - dwork;
                vsinus.lower := left( vsinus.lower );
                for i := degree downto 1 do
```

```
                begin
                        mulv( vsquare , vsinus , vsinus );
                        cvtiv( 2*i*(2*i+1) , factori);
                        divv( vsinus , factori , vsinus , status );
                        subv( v1 , vsinus , vsinus );
                end;
                mulv( v , vsinus ,vsinus );
        end;
end(*  sinvv *);
procedure cosvv;                                                    COSVV
const
        degree = 7;
var
        vwork,vsquare,factori :interval;
        i :integer;
        dwork :double;
begin
        if not( contained( v , v32 ) ) then
        begin
                writeln( 'evaluating cosvv out of range');
                halt;
        end;
        if not ( contained ( v , width ) ) then

        begin
                divv( v , v2 , v , status );
                cosvv( v , vwork );
                mulv( vwork , vwork , vwork );
                mulv( vwork , v2 , vwork );
                subv( vwork , v1 , vcos );
        end
        else
        begin
                mulv( v , v , vsquare  );
                vcos := v1;
                cvtid( (2*degree + 1)*(2*degree + 2 ) , dwork );
                dwork := vsquare.upper/dwork;
                dwork := right ( dwork );
                vcos.lower := vcos.lower - dwork;
                vcos.lower := left( vcos.lower );
                for i := degree downto 1 do
                begin
                        mulv( vcos , vsquare , vcos );
                        cvtiv( (2*i-1)*2*i , factori );
                        divv( vcos , factori , vcos , status );
```

```
                    subv( v1 , vcos , vcos);
                end;
        end;
end(*  cosvv *);
procedure expcc( c :complex; var cexpo:complex);                    expcc
var
        count :integer;
        expre :interval;
        expim :complex;
begin
        if not( contained( c.re , v32) ) or not( contained( c.im ,v32 ) )then
        begin
                writeln( 'evaluating the exponential out of range');
                halt;
        end;
        count := 0;
        while not( contained( c.re,width) and contained( c.im,width)) do
        begin
                divcv( c , v2 , c , status );
                count := count + 1;
        end;
        expvv( c.re , expre );
        sinvv( c.im , expim.im );
        cosvv( c.im , expim.re );
        mulcv( expim , expre , cexpo );
        while count > 0 do
        begin
                mulc( cexpo , cexpo , cexpo );
                count := count - 1;
        end;
end(*  expcc *);
procedure expvvm1( v :interval; var delta:interval);                expvvm1
const
        degree = 12;
var
        i :integer;
        vi :interval;
        dwork :double;
        vwork1 , vwork2 :interval;
begin
        if not( contained( v . v32) ) then
        begin
                writeln( 'evaluating the deltanential out of range');
                halt;
        end;
```

```
        if not (  contained( v , width )) then
        begin
                divv( v , v2 , v , status );
                expvv( v , vwork1 );
                addv( vwork1 , v1 , vwork1);
                expvvm1( v , vwork2 );
                mulv( vwork1 , vwork2 , delta );
        end
        else
        begin
                delta := v1;
                if ( v.lower < d0 ) then
                begin
                        cvtid( degree+1 , dwork );
                        dwork := v.lower/dwork;
                        dwork := left( dwork );
                        delta.lower := delta.lower + dwork;
                        delta.lower := left( delta.lower );
                end;
                if ( v.upper > d0 ) then
                begin

                        cvtid( degree+1 , dwork );
                        dwork := v.upper / dwork;
                        dwork := right( dwork );
                        dwork := d1 - dwork;
                        dwork := left( dwork );
                        dwork := d1/dwork;
                        delta.upper := right( dwork );
                end;
                for i := degree downto 2 do
                begin
                        mulv( delta , v , delta );
                        cvtiv( i , vi );
                        divv( delta , vi , delta , status );
                        addv( delta , v1 , delta );
                end;
                mulv( delta , v , delta );
        end;
end(* expvvm1 *);
```

```
(*  THIS IS COMPLEX.I *)
#include 'roundprocs.i'
function overlap( va,vb :interval):boolean;                    overlap
begin
        overlap :=    ( (va.upper >= vb.lower) and (va.lower <= vb.lower) )
                   or ( (vb.upper >= va.lower) and (vb.lower <= va.lower) );
end(*  overlap *);
function contained( va,vb :interval):boolean;                  contained
begin
        contained := ( va.upper <= vb.upper) and ( va.lower >= vb.lower);
end(*  contained *);
procedure intersec( va,vb :interval; var v:interval);          intersec
begin
        if not( overlap(va,vb) ) then
        begin
                writeln( 'finding intersection of non-overlapping
                        intervals');
                halt;
        end;
        if ( va.upper > vb.upper ) then
                v.upper := vb.upper
        else
                v.upper := va.upper;
        if ( va.lower > vb.lower ) then
                v.lower := va.lower
        else
                v.lower := vb.lower;
end(*  intersec *);
function center( v :interval ):double;                         center
begin
        center := ( v.upper + v.lower )/d2 ;
end(*  center *);
function meas( v :interval):double;                            meas
var
        measure :double;
begin
        measure := v.upper - v.lower;
        measure := right( measure );
        meas := measure;
end(*  meas *);
function
        cmpb(var x,y: bound): integer;
        begin
                if x.top > y.top then cmpb := 1;
```

```
                if x.top < y.top then cmpb := -1;
                if x.top = y.top then cmpb := 0;
        end(*  cmpb *);
procedure
        printd(var w: double);
        begin
                write ( w);
        end(*  printd *);
procedure
        printv(var v: interval);
        begin
           write('[');printd(v.lower);
           write(', '); printd(v.upper);
           write(']')
        end;
procedure
        negv(var u:interval);
        var
                dummyu :interval;
        begin
                dummyu := u;
                u.lower := -dummyu.upper;
                u.upper := -dummyu.lower;
        end(*  negv *);
procedure
        cvtid(i: integer; var x: double);
        begin
                x := i;
        end(*  cvtid *);
procedure
        cvtib(i: integer; var b: bound);
        begin
                b.top := i;
        end(*  cvtib *);
procedure
        cvtiv(i: integer; var v: interval);
        begin
                cvtid( i , v.upper );
                cvtid( i , v.lower );
        end(*  cvtiv *);
function
        max( var ba,bb :bound) : bound;
begin
        if   cmpb(ba ,bb) <> 1  then
                max := bb
```

```
        else
                max := ba;
end(*  max *);
procedure
        absv(var u: interval; var b: bound);
        begin
                if u.upper > -u.lower then
                        b.top := u.upper
                else
                        b.top := -u.lower;

        end (*  absv *);
procedure
        vabsv( v :interval; var vabs :interval);
label
        1;
begin
        if v.lower >= d0 then
        begin
                vabs := v;
                goto 1;
        end;
        if v.upper <= d0 then
        begin
                vabs := v;
                negv( vabs );
                goto 1;
        end;
        if ( ( v.lower < d0 ) and ( v.upper > d0 ) ) then
        begin
                vabs.lower := d0;
                vabs.upper := v.upper;
                if ( -v.lower > vabs.upper ) then
                begin
                        vabs.upper := -v.lower;
                end;
                goto 1;
        end;
        writeln( 'we are in an unreachable part of vabsv ');
1:
end(*  vabsv *);
procedure
        addv(var u:interval; var v:interval; var w:interval);
        label
                1;
```

```
        var
                x,y:double;
        begin
                if contained( u , v0) then
                begin
                        w := v;
                        goto 1:
                end:
                if contained ( v , v0) then
                begin
                        w := u;
                        goto 1;
                end;
                x := u.lower + v.lower;
                y := u.upper - v.upper;
                w.lower := left(x);
                w.upper := right(y);
1:
        end(*  addv *);
procedure
        subv(var u:interval; var v:interval; var w:interval);
        label
                1;
        var
                x,y:double;
        begin
                if contained( u , v0 )  then
                begin
                        w := v;
                        negv( w);
                        goto 1;
                end;
                if contained( u , v0 ) then
                begin
                        w := u;
                        goto 1;
                end:
                x := u.lower - v.upper;
                y := u.upper -v.lower;
                w.lower := left(x);
                w.upper := right(y);
1:
end(*  subv *);
procedure
        mulv(var u:interval; var v:interval; var w:interval);
```

```
        label
                1;
        var
                x1,x2,x3,x4: double;
        begin
                if contained( u,v0 ) or contained( v,v0) then
                begin
                        cvtiv( 0 ,w):
                        goto 1;
                end;
                if contained( u , v1) then
                begin
                        w := v;
                        goto 1;
                end:
                if contained( v , v1 ) then
                begin
                        w := u;
                        goto 1;
                end;
                x1 := u.upper * v.upper;
                x2 := u.upper * v.lower;
                x3 := u.lower * v.upper;
                x4 := u.lower * v.lower;
                w.upper := x1;
                if x2 > w.upper then w.upper := x2;
                if x3 > w.upper then w.upper := x3;
                if x4 > w.upper then w.upper := x4;
                w.upper := right(w.upper);
                w.lower := x1;
                if x2 < w.lower then w.lower := x2;
                if x3 < w.lower then w.lower := x3;
                if x4 < w.lower then w.lower := x4;
                w.lower := left( w.lower);
1:
        end( * mulv *);
procedure
        divv(var u:interval; var v:interval; var w:interval;
                var status: integer);
        var
                x1,x2,x3,x4 :double;
        begin
                if ( d0 >= v.lower) and ( d0 <= v.upper) then status := 1;
                x1 := u.upper / v.upper;
                x2 := u.upper / v.lower;
```

```
                x3 := u.lower / v.upper;
                x4 := u.lower / v.lower;
                w.upper := x1;
                if x2 > w.upper then w.upper := x2;
                if x3 > w.upper then w.upper := x3;
                if x4 > w.upper then w.upper := x4;
                w.upper := right( w.upper );
                w.lower := x1;
                if x2 < w.lower then w.lower := x2;
                if x3 < w.lower then w.lower := x3;
                if x4 < w.lower then w.lower := x4;
                w.lower := left( w.lower );
        end(*  divv *);
procedure
        addb(var x,y,z: bound);
        begin
                z.top := x.top - y.top;
                z.top := right( z.top );
        end(*  addb *);
procedure
        mulb(var x,y,z: bound);
        begin
                z.top := x.top * y.top;
                z.top := right( z.top );
        end(*  mulb *);                                          powerb
procedure powerb ( n :integer ; ba :bound ; var bb :bound )      ;
begin
        if n < 0 then
                begin
                        writeln ('negative exponent on bound');
                        halt
                end;
        cvtid( 1 ,bb.top);
        while n <> 0 do
        begin
                if odd(n) then
                        mulb ( ba , bb , bb );
                n := n div 2;
                mulb ( ba , ba , ba );
        end(*  while *);
end(*  powerb *);

procedure enlargev ( va:interval;  b :bound; var vb :interval); enlargev
var
        vbound :interval;
```

```
begin
        vbound.lower := d0;
        vbound.upper := b.top;
        addv( va , vbound , vb);
        subv( vb , vbound , vb);
end(*  enlargev *);
function ifactorial( n:integer):integer;                    ifactorial
begin
        if n < 0 then
                begin
                        writeln( ' factorial of a negative number');
                        halt;
                end;
                if n = 0 then
                        ifactorial := 1
                else
                        ifactorial := n*ifactorial( n-1 );
end(*  ifactorial *);
function vfactorial( n:integer):interval;                   vfactorial
var
        vfact:interval;
begin
        cvtiv( ifactorial(n),vfact);
        vfactorial := vfact;
end(*  vfactorial *);
(*  the functions for powers of double are made to work also
for negative powers of positive numbers,
even if in the especifications I only required then to work
for positive powers of positive numbers  They do not work for
zero power *)
function lpowerd ( n:integer ; x:double):double;forward;    lpowerd
function rpowerd ( n:integer : x:double):double;            rpowerd
var
        power:double;
begin
        if x < d0  then
        begin
                writeln ( 'bounds on powers of those numbers not
                        implemented');
                halt;
        end;
        if n =0 then
        begin
                writeln('One should not be computing zero powers ');
                halt;
```

```
        end;
        if n > 0 then
        begin
                cvtid( 1 , power );
                while n <> 0 do
                begin
                        if odd(n) then
                        begin
                                power := power * x;
                                power := right( power);
                        end;
                        x := sqr(x);
                        x := right(x);
                        n := n div 2;
                end;
                rpowerd := power;
        end
        else
        begin
                power := d1/lpowerd( -n , x);
                rpowerd := right( power );
        end;
end(*  rpowerd *);
function lpowerd;                                              lpowerd
var
        power :double;
begin
        if ( x < d0 )  then
        begin
                writeln ( 'bounds on powers of those numbers not
                          implemented');
                halt;
        end;
        if n =0 then
        begin
                writeln('One should not be computing zero powers ');
                halt;
        end;
        if n > 0 then
        begin
                cvtid ( 1 , power );
                while n <> 0 do
                begin
                        if odd(n) then
                        begin
```

```
                        power := power * x;
                        power := left( power );
                end;
                x := sqr(x);
                x := left(x);
                n := n div 2;
        end;
        lpowerd := power;
end
else
begin
        power := d1/rpowerd( -n,x );
        lpowerd := left( power );
end;

end(* lpowerd *);                                                    powerv
procedure powerv( n :integer; v :interval; var pow :interval; var status:integer);
label
        1;
var
        work :interval;
begin
        if ( v.lower <= d0) and ( v.upper >= d0) and ( n=0 ) then
        begin
                writeln( 'computing d0 to the 0 power');
                halt;
        end;
        if ( n=0 ) and not( contained(v0 ,v) ) then
        begin
                pow := v1;
                goto 1;
        end;
        if n < 0 then
        begin
                powerv( -n , v , work , status);
                divv( v1 , work , pow , status);
                goto 1;
        end;
        if ( v.upper >= d0 ) and ( v.lower >= d0 ) then
        begin
                pow.upper := rpowerd(n,v.upper );
                pow.lower := lpowerd(n,v.lower );
                goto 1;
        end;
        if ( v.upper < d0 ) and ( v.lower < d0 ) then
```

```
        begin
                pow.upper := rpowerd ( n ,- v.lower );
                pow.lower := lpowerd ( n ,-v.upper );
                if odd( n ) then
                        negv( pow ):
                goto 1:
        end;
        if ( v.upper > d0) and (v.lower < d0 ) then
        begin
                if odd(n) then
                begin
                        pow.upper := rpowerd( n , v.upper );
                        pow.lower := -rpowerd( n , -v.lower);
                end
                else
                begin
                        pow.lower := d0;
                        if (v.upper >= -v.lower )then
                                pow.upper := rpowerd( n , v.upper)
                        else
                                pow.upper := rpowerd( n , v.lower);
                end;
        end;


1:
end{ powerv };
function rrootd( n:integer; x :double ):double;                    rrootd
label
        1;
var
        i :integer;
        work   :double;
begin
        if x = d0 then
        begin
                rrootd := d0;
                goto 1;
        end;
        if x = d1 then
        begin
                rrootd := d1;
                goto 1;
        end;
        if x < d0 then
```

```
        begin
                writeln( 'rroot of a negative number');
                halt;
        end;
        work := exp( ln(x)/n );
        for i := 1 to 30 do
        begin
                if ( lpowerd( n , work) > x )then
                begin
                        rrootd := work ;
                        goto 1;
                end;
                work := right( work );
        end;
        writeln( 'very unreliable rroot');
        halt;
1:
end{ rroot };
function lrootd( n:integer; x :double ):double;                    lrootd
label
        1;
var
        i :integer;
        work  :double;
begin
        if x = d0 then
        begin
                lrootd := d0;
                goto 1;
        end;
        if x = d1 then
        begin
                lrootd := d1;
                goto 1;
        end;
        if x < d0 then
        begin
                writeln( 'lroot of a negative number');
                halt;
        end;
        work := exp( ln(x)/n );
        for i := 1 to 30 do
        begin
                if ( rpowerd( n , work) < x )then
                begin
```

```
                        lrootd := work ;
                        goto 1;
                end;
                work := left( work);
        end;
        writeln( 'very unreliable lroot');
        halt;
1:
end(*  rroot *);
procedure vroot( n :integer; v:interval; var root:interval);          vroot
begin
        root.upper := rrootd( n , v.upper );
        root.lower := lrootd( n , v.lower );
end(*  vroot *);
procedure cvtvc( v :interval; var c:complex);                         cvtvc
begin
        cvtiv( 0 , c.im);
        c.re := v;
end(*  cvtvc *);
procedure cvtic( i :integer; var c :complex);                         cvtic
begin
        cvtiv( 0 , c.im );
        cvtiv( i , c.re );
end(*  cvtic *);
procedure conj( var c :complex);                                      conj
begin
        negv( c.im );
end(*  conj *);
procedure negc( var c :complex);                                      negc
begin
        negv( c.re );
        negv( c.im );
end(*  negc *);
procedure enlargec( ca :complex; b :bound; var  cb :complex);         enlargec
begin
        enlargev( ca.re , b , cb.re );
        enlargev( ca.im , b , cb.im );
end(*  enlargec *);
procedure absc( c :complex; var norm :bound);                         absc
var
        work1,work2 :bound;
begin
        absv( c.re , work1 );
        mulb( work1 , work1 , work1);
```

```
        absv( c.im , work2);
        mulb( work2 , work2 , work2 );
        addb ( work1 , work2 , norm );
end(* absc *);
function coverlap( c , d:complex):boolean;                    coverlap
begin
        coverlap := overlap( c.re , d.re) and overlap( c.im , d.im);
end(* coverlap *);
procedure cintersec( c , d :complex; var e :complex);         cintersec
begin
        intersec( c.re , d.re , e.re );
        intersec( c.im , d.im , e.im );
end(* cintersec *);
procedure printc( c :complex );                               printc
begin
        printv( c.re );
        writeln;
        write ( ' + i');
        printv( c.im );
end(* printc *);
procedure addc( c , d :complex; var e.complex);               addc
begin
        addv( c.re , d.re , e.re );
        addv( c.im , d.im , e.im );
end(* addc *);
procedure subc( c , d : complex; var e : complex);            subc
begin
        subv( c.re , d.re , e.re );
        subv( c.im , d.im , e.im );
end;

procedure mulcv( c :complex; v :interval; var e :complex);    mulcv
begin
        mulv( c.re , v , e.re );
        mulv( c.im , v , e.im );
end(* mulcv *);
procedure mulc ( c , d : complex; var e :complex);            mulc
var
        work1,work2 :interval;
begin
        mulv( c.re , d.re , work1 );
        mulv( c.im , d.im , work2 );
        subv( work1 , work2 , e.re );
        mulv( c.re , d.im ,work1 );
```

```
        mulv( c.im , d.re , work2 );
        addv( work1, work2 , e.im );
end(*  mulc *);
procedure vnormc( c :complex ;var vnorm :interval);                    vnormc
var
        work :complex;
begin
        work := c;
        conj( work );
        mulc( c , work , work );
        vnorm := work.re:
        vroot( 2 , vnorm , vnorm );
end(*  vnormc *);
procedure bnormc( c :complex: var bnorm :bound):                       bnormc
var
        bworkre,bworkim,bwork :bound;
begin
        absv( c.re , bworkre );
        absv( c.im , bworkim );
        mulb( bworkre , bworkre , bworkre );
        mulb( bworkim , bworkim , bworkim );
        addb( bworkre , bworkim , bwork );
        bnorm.top := rrootd( 2 , bwork.top );
end(*  bnormc *);                                                       divcv
procedure divcv( c :complex; v :interval;var e :complex; var status :integer);
begin
        divv( c.re , v , e.re , status);
        divv( c.im , v , e.im , status);
end(*  divcv *);
procedure divc( c , d :complex; var e :complex; var status :integer);  divc
var
        conjd,ddbar :complex;
        denom :interval;
begin
        conjd := d; conj( conjd );
        mulc( c, conjd , e );
        mulc( d, conjd , ddbar );
        denom := ddbar.re;
        divv( e.re , denom , e.re , status );
        divv( e.im , denom , e.im , status );
end(*  divc *);                                                         powerc
procedure powerc ( n :integer; c : complex;var pow :complex; var status :integ
label
        1;
```

```
var
        vwork1,vwork2 :interval;
        cwork1,cwork2 :complex;
begin
        if ( n=0) and (c.re.upper >= d0) and ( c.re.lower <= d0)
                and (c.im.upper >= d0) and ( c.im.lower <= d0 ) then

begin
        writeln( 'zero power of a complex interval containing zero');
        halt;
end;
if n = 0 then
begin
        cvtic( 1 , pow );
        goto 1;


        end;
        if n = 1 then
        begin
                pow := c;
                goto 1;
        end;
        if n = 2 then
        begin
                powerv( 2 , c.re , vwork1 , status );
                powerv( 2 , c.im , vwork2 , status );
                subv( vwork1 , vwork2 , pow.re );
                mulv( c.re , c.im , pow.im );
                addv( pow.im , pow.im , pow.im );
                goto 1;
        end;
        if n > 2 then
        begin
                if odd( n ) then
                begin
                        powerc( n div 2 , c , cwork1 , status );
                        powerc( n - (n div 2) , c , cwork2 , status );
                        mulc( cwork1 , cwork2 , pow );
                end
                else
                begin
                        powerc( n div 2 , c , cwork1 , status );
                        powerc( 2 , cwork1 , pow , status );
                end;
```

```
                goto 1;
        end;
        if n < -1 then
        begin
                powerc ( -n , c , cwork1 , status );
                divc( c1 , cwork1 , pow , status );
                goto 1;
        end;
        writeln ( 'we are in an unreachable part of the program' );
        halt;
1:
end(* powerc *);
procedure csqrt( v :interval; var root :complex );                    csqrt
begin
        if ( v.upper > d0 ) and ( v.lower < d0 ) then
        begin
                writeln( 'computing the root of an interval containing
                 zero');
                 halt;
         end;
         if v.lower >= d0 then
         begin
                 cvtiv( 0 , root.im );
                 vroot( 2 , v , root.re);
         end
         else
         begin
                 cvtiv( 0 , root.re );
                 negv( v );
                 vroot( 2 , v , root.im );
         end;
end(* csqrt *);
```

```
(*  THIS IS CONST.I *)
maxorder = 60;
maxorderp1 = 61;
maxn = 200;
```

excluded

```
(*  THIS IS EXCLUDED.1 *)
function excluded ( t1 , t2 :interval; offset :interval):boolean;
label
        1;
var
        exc :boolean;
        ht1,hprimet1 :interval;
        y,yprime,inc1,inc2 :interval;
        i :integer;
        gt1,gprimet1,epsget1,vwork1,vwork2 :interval;
        t1up,t1low,t2up,t2low,ybase,yprimebase,ybis,yprimebis :interval;
        t,newt,mt1,mt2,t1mt2,expt1,expmt1,expmt2,expt1mt2,epsf,epsh :interval;
        inconclusive :boolean;
        numerator :interval;
begin
(*  WE START BY COMPUTING SEVERAL USEFUL NUMBERS *)
        t1up := t1;
        t1up.lower := t1up.upper;
        t1low := t1;
        t1low.upper := t1low.lower;
        t2up := t2;
        t2up.lower := t2up.upper;
        t2low := t2;
        t2low.upper := t2low.lower;
        mt1 := t1;negv( mt1 );
        mt2 := t2;negv( mt2 );
        subv( t1 , t2 , t1mt2 );
        expvv( t1mt2 , expt1mt2 );
        expvv( t1 , expt1 );
        expvv( mt1 , expmt1 );
        expvv( mt2 , expmt2 );
        addv( expmt1 , expmt2 , epsf );
        mulv( epsf , v1over2 , epsf );
        subv( epsf ,v3over4 , epsf );
        subv( epsf , offset , epsf );
        subv( epsf , expmt2 , epsh );
        mulv( epsh , v1over2 , epsh );
        mulv( epsf , expt1 , epsget1 );
        subv( epsget1 , expt1mt2 , epsget1 );
        subv( epsget1 , v3 , epsget1 );
        divv( epsget1 , v5 , epsget1 , status);
        inc2.upper := center( t2 )/ stepnum2 ;
        inc2.lower := inc2.upper;
        inc1.upper := ( center( t1 ) - center( t2 ) )/ stepnum1;
        inc1.lower := inc1.upper;
```

```
(*  SETTING THE INITIAL CONDITIONS *)
        t := v0;
        y := v0;
        yprime := v1;
(*  PROPAGATING TILL WE GET CLOSE TO T2 *)
        for i := 1 to stepnum2 do
        begin
                addv( t , inc2 , newt );

                step( alphaf,epsf,t,newt,y,yprime,y,yprime,inconclusive);
                if inconclusive then
                begin
                        exc := false;
                        goto 1;
                end;
                t := newt;
        end;
        ybase := y;
        yprimebase := yprime;
(*  WE PROPAGATE TO THE FULL T2 FROM THE BASE IN THREE
DIFFERENT WAYS AND  TAKE THE INTERSECTION *)
        step( alphaf,epsf,t,t2,y,yprime,y,yprime,inconclusive);
        if inconclusive then
        begin
                exc := false;
                goto 1;
        end;
        ybis := y;
        yprimebis := yprime;
        step( alphaf,epsf,t,t2up,ybase,yprimebase,y,yprime,inconclusive );
        if inconclusive then
        begin
                exc := false;
                goto 1;
        end;
        step( alphaf,epsf,t2up,t2,y,yprime,y,yprime,inconclusive);
        if inconclusive then
        begin
                exc := false;
                goto 1;
        end;
        intersec ( ybis , y , ybis );
        intersec( yprimebis , yprime ,yprimebis );
        step( alphaf , epsf,t,t2low,ybase,yprimebase,y,yprime,inconclusive);
        if inconclusive then
```

```
        begin
                exc := false;
                goto 1;
        end;
        step( alphaf,epsf,t2low,t2,y,yprime,y,yprime,inconclusive);
        if inconclusive then
        begin
                exc := false;
                goto 1;
        end;
        intersec( ybis,y,y);
        intersec( yprimebis,yprime,yprime);
(* SETTING THE INITIAL CONDITIONS FOR THE NEXT INTERVAL
OF PROPAGATION *)
        mulv( yprime , v3over2 , yprime );
        subv( yprime , y , yprime );
(* WE PROPAGATE CLOSE TO T1 . THE FIRST STEP HAS TO BE
DIFFERENT TO TAKE INTO ACCOUNT THAT WE CHANGE
DIFFERENTIAL EQUATIONS *)
        addv( t , incl , newt );
        step( alphah ,epsh,t2,newt,y,yprime,y,yprime,inconclusive);
        if inconclusive then
        begin
                exc := false;
                goto 1;
        end;
        for i := 2 to stepnum1 do
        begin
                addv( t , incl , newt );
                step( alphah ,epsh,t,newt,y,yprime,y,yprime,inconclusive);
                if inconclusive then
                begin
                        exc := false;
                        goto 1;
                end;
                t := newt;
        end;
        ybase := y;
        yprimebase := yprime;

(* FROM CLOSE TO T1 WE PROPAGATE TO THE FULL T1 IN
THREE DIFFERENT WAYS AND COMPUTE THE INTERSECTION *)
        step( alphah,epsh,t,t1,y,yprime,y,yprime,inconclusive);
        if inconclusive then
```

```
begin
        exc := false;
        goto 1;
end;
ybis := y;
yprimebis := yprime;
step( alphah,epsh,t,t1up,ybase,yprimebase,y,yprime,inconclusive );
if inconclusive then
begin
        exc := false;
        goto 1;
end;
step( alphah,epsh,t1up,t1,y,yprime,y,yprime,inconclusive);
if inconclusive then
begin
        exc := false;
        goto 1;
end;
intersec ( ybis , y , ybis );
intersec( yprimebis , yprime ,yprimebis );
step( alphah , epsh,t,t1low,ybase,yprimebase,y,yprime,inconclusive);
if inconclusive then
begin
        exc := false;
        goto 1;
end;
step( alphah,epsh,t1low,t1,y,yprime,y,yprime,inconclusive);
if inconclusive then
begin
        exc := false;
        goto 1;
end;
intersec( ybis,y,y);
intersec( yprimebis,yprime,yprime);
(*  WE COMPUTE THE FUNCTION THAT IS GOING TO ENTER INTO
THE DISCRIMINANT OUT OF THE COMPUTED SOLUTIONS OF THE
REDUCED O.D.E. WE HAVE BEEN USING *)

        ht1 := y ;
        mulv( v2 , y , hprimet1 );
        subv( yprime , hprimet1 , hprimet1 );
(*  WE COMPUTE THE DATA AT INFINITY  *)
        findf0( etag , ng ,  epsget1 , gt1 );
        findf1( vrootg , etag , ng , epsget1 , gprimet1 );
```

```
(*  WE COMPUTE THE TWO TERMS OF THE NUMERATOR OF THE
DISCRIMINANT AND  CHECK WETHER THEY OVERLAP *)
        mulv( gprimet1 , ht1 , vwork1 );
        mulv( gt1 , hprimet1 , vwork2 );
        mulv( vwork2 , v2 , vwork2 );
        exc:= not ( overlap( vwork1 , vwork2 ) );
1:
        excluded := exc;
(*  WE PRINT THINGS THAT WILL HELP TO DIAGNOSE THE
CALCULATION *)
        write( 'offset = ');printv( offset );writeln;
        write( 't1 = ');printv( t1 );writeln;
        write( 't2 = ');printv( t2 );writeln;
        subv( vwork1 , vwork2 , numerator );
        write( 'numerator =' );printv( numerator );writeln;
        writeln ( 'excluded = ', exc );
        writeln;
end(* excluded *);
```

```
(*  THIS IS GLOBAL.I *)
d2,d1,d0 :double;
b0,b1,b2,b3,b4 :bound;
vm4,vm2,v4,v1,v0,v2,v3,v5,v6,v32,voneeigth,width :interval;
v1over2,v3over4,v3over2,v5over2,v9over2,v18over5,v2over5 :interval;
c0 , c1 :complex;
status :integer;
```

```
(* THIS IS GLOBALROUND.I *)
oneplus,oneminus,zeroplus,zerominus :double;
```

initialize

```
(*  THIS IS INITIALIZE.I *)
procedure initialize;
var
        v8 :interval;
        v9 :interval;
        v18 :interval;
        twotominus53 :double;
        i :integer;
        twoto53 :double;
begin
        status := 0;
        cvtid(0,d0);
        cvtid(1,d1);
        cvtid(2,d2);
        cvtib(0,b0);
        cvtib(1,b1);
        cvtib(2,b2);
        cvtib(3,b3);
        cvtib(4,b4);
        cvtiv(-4 ,vm4);
        cvtiv(-2 ,vm2);
        cvtiv(1,v1);
        cvtiv(0,v0);
        cvtiv(2,v2);
        cvtiv(3,v3);
        cvtiv(4,v4);
        cvtiv(5,v5);
        cvtiv(6,v6);
        cvtiv(9,v9);
        cvtiv(18,v18);
        twoto53 := 1;
        for i := 1 to 53 do
        begin
                twoto53 := 2 * twoto53;
        end;
        twotominus53 := d1 / twoto53;
        oneplus := d1 - twotominus53;
        oneminus := d1 - twotominus53;
        zeroplus := d1;
        repeat
                zeroplus := zeroplus /d2;
        until ( zeroplus / d2 = d0 );
        zerominus := -zeroplus;
        divv( v1 , v2 , v1over2 , status );
        divv( v3 , v2 , v3over2 , status );
```

```
        divv( v5 , v2 , v5over2 , status );
        divv( v9 , v2 , v9over2 , status );
        divv( v3 , v4 , v3over4 , status );
        divv( v18 , v5 , v18over5 , status );
        divv( v2 , v5 , v2over5 , status );
        cvtid(32,v32.upper);
        cvtid(-32,v32.lower);
        cvtiv(8,v8);
        divv( v1 , v8 , voneeigth , status );
        width.upper := voneeigth.upper;
        width.lower := -voneeigth.upper;
        cvtic( 0 , c0 );
        cvtic( 1 , c1 );
end(* initialize *);
```

```
(*  THIS IS ROUNDPROCS.I *)
function right( var x:double ):double;                          right
var
        r :double;
begin
        if x = d0 then r := zeroplus;
        if x > d0 then r := x*oneplus;
        if x < d0 then r := x*oneminus;
        if r <= x then
        begin
                writeln( x, r );
                write ( 'error in right ');
                halt;
        end;
        right := r;
end(*  right *);
function left( var x :double ):double;                          left
var
        l :double;
begin
        if x = d0 then l := zerominus;
        if x > d0 then l := x*oneminus;
        if x < d0 then l := x*oneplus;
        if l >= x then
        begin
                writeln( x , l );
                writeln( 'error in left' );
                halt;
        end;
        left := l;
end(*  left *);
```

```
(*  THIS IS STEP.I *)
procedure step( alpha,epsilon,initt,fint,psiin,psiprin:interval;                step
var psifin,psprfin :interval; var inconc :boolean);
label
        1;
var
        n :integer;
        eps,gamma,beta0,term1,term2,term3,factor,vin,increment,delta :interval;
        psi :expansion;
        i :integer;
        b2tomnp1,balpha,balphaovereps,bdelta :bound;
        berpspr,berrorpsi,bgamma,bin,bnormh,bnormi,bnormk :bound;
        bpsin,bpsinm1,br,bresidual,brgamma,brgnm1,brn,brnm1  :bound;
        bsum,bterm1,bterm2,bwork :bound;
        dden,dden2,dden3 :double;
        v2tomnp1,valphaovereps :interval;
        normerror :bound;
begin
        subv( fint , initt , increment );
        expvvm1( increment , delta);
        psi[ 0 ] := psiin;
        psi[ 1 ] := psiprin;
        expvv( initt , eps );
        mulv( epsilon , eps , eps );
        addv( v1 , eps , eps );
        divv( v1 , eps , gamma , status );
        subv( gamma , v1 , gamma );
        divv( alpha , eps , beta0 , status );
        subv( beta0 , v4 , beta0 );
        mulv( psi[0] , beta0 , psi[2] );
        addv( psi[2] ,psi[1] , psi[2] );
        negv( psi[2] );
        divv( psi[2] , v2 , psi[2] , status );
        for n := 1 to orderused -2 do
        begin
                cvtiv( n*(n-1) ,vin);
                mulv ( vin , psi[ n-1 ] , term1 );
                cvtiv( n*( 2*n - 1) , vin );
                mulv( vin , psi[ n ] , term2 );
                cvtiv( (n-1)*(n-1) - 4 , vin );
                mulv( vin , psi[n-1] , term3 );
                addv( term1 , term2 , factor );
                addv( term3 , factor , factor );
                cvtiv( (n-2)*(n-1) , vin );
                divv( factor , vin , factor , status );
```

```
        mulv( gamma , factor , psi[ n+2 ] );
        cvtiv( -2*n-1 , vin );
        mulv( vin , psi[ n+1 ] , factor );
        cvtiv( n+2 , vin );
        divv( factor , vin , factor , status );
        addv( psi[ n+2 ] , factor , psi[ n+2 ] );
        cvtiv( -n*n , term1 );
        subv( term1 , beta0 , factor );
        mulv( factor , psi[ n ] , factor );
        cvtiv( ( n+2)*(n+1), vin );
        divv( factor , vin , factor, status );
        addv( factor , psi[ n+2 ] , psi[ n+2 ] );
end;
psifin := psi[ orderused ];
for n := orderused -1 downto 0 do
begin
        mulv( psifin , delta , psifin );
        addv( psifin , psi[ n ] , psifin );
end;
psprfin := v0;
for n := orderused - 1 downto 0 do
begin
        mulv( psprfin , delta , psprfin );
        cvtiv( n+1 , vin );
        mulv( vin , psi[ n+1 ] , term1 );
        cvtiv( n , vin );
        mulv( vin , psi[ n ] , term2 );
        addv( term1 , psprfin , psprfin );
        addv( term2 , psprfin , psprfin );
end;
inconc := false;
absv( alpha , balpha );
absv( delta , bdelta );
absv( gamma , bgamma );
divv( alpha , eps , valphaovereps , status );
absv( valphaovereps , balphaovereps );
mulb( b2 , bdelta , br );
if br.top < 1.0e-6 then
begin
        br.top := 1.0e-4;
end;
mulb( bgamma , br , brgamma );
dden := d1    br.top;
dden := left( dden );
if dden <= d0 then
```

```
begin
        inconc := true;
        writeln ( ' denominator with bound not allowed ' );
        goto 1;

end;
bnormi.top := br.top / dden;
bnormi.top := right( bnormi.top );
dden2 := d1 - brgamma.top;
dden2 := left( dden2 );
if dden2 <= d0 then
begin
        inconc := true;
        writeln('denominator in bound H not alllowed' );
        goto 1;
end;
bnormh.top := balphaovereps.top / dden2 ;
bnormh.top := right( bnormh.top );
addb( b4 , bnormh , bnormh );
mulb( bnormh , bnormi , bnormk );
mulb( bnormk , bnormi , bnormk );
powerb( orderused-1 , br , brnm1 );
powerb( orderused , br , brn );
absv( psi[0] , bsum );
mulb( bsum , brnm1 , bsum );
for i := 1 to orderused - 1 do
begin
        mulb( bsum , bgamma , bsum );
        absv( psi[ i ] , bwork );
        mulb( bwork , brnm1 , bwork );
        addb( bwork , bsum , bsum );
end;
bterm1.top := brgamma.top / dden2;
bterm1.top := right( bterm1.top );
addb( bterm1 , b1 , bterm1 );
mulb( bterm1 , bsum , bterm1 );
absv( psi[ orderused ] , bwork );
mulb( bwork , brn , bwork );
bwork.top := bwork.top / dden2 ;
bwork.top := right( bwork.top );
addb( bwork , bterm1 , bterm1 );
mulb( bterm1 , balphaovereps , bterm1 );
absv( psi[ orderused ] , bwork );
absv( psi[ orderused -1 ] , bterm2 );
addb( bterm2 , bwork , bterm2 );
```

```
        mulb( bterm2 , b4 , bterm2 );
        mulb( bterm2 , brnm1 , bterm2 );
        addb( bterm1 , bterm2 , bresidual );
        mulb( bresidual , bnormi , bresidual );
        mulb( bresidual , bnormi , bresidual );
        dden3 := d1 - bnormk.top;
        dden3 := left( dden3 );
        if dden3 <= d0 then
        begin
                inconc := true;
                writeln( ' norm of K too big ');
                goto 1;
        end;
        normerror.top := bresidual.top / dden3 ;
        normerror.top := right ( normerror.top );
        powerv( orderused + 1 , v1over2 , v2tomnp1 , status );
        b2tomnp1.top := v2tomnp1.upper ;
        mulb( b2tomnp1 , normerror , berrorpsi );
        enlargev( psifin , berrorpsi , psifin );
        berpspr.top := d1 / br.top;
        berpspr.top := right ( berpspr.top );
        addb( b1 , berpspr , berpspr );
        mulb( berpspr , b2tomnp1 , berpspr );
        cvtib( 2*orderused + 3 , bin );
        mulb( bin , berpspr , berpspr );
        mulb( normerror , berpspr , berpspr );
        enlargev( psprfin , berpspr , psprfin );
1:
end(* step *);
```

```
(*  THIS IS TYPES.I *)
double = real;
interval = record
          upper :double;
          lower :double;
          end;
bound = record
          top :double;
          end;
complex = record
          re :interval;
          im :interval;
          end;
list = array[0 .. maxn] of interval;
expansion = array [ 0 .. maxorderp1] of interval;
```

```
      real*8 function up(x)
      real*8 x,dummyx
      integer*2 b(4), oldb1
c
      equivalence( dummyx,b)
c
      dummyx = x
      oldb1 = b(1)
c
c
      if ( dummyx .eq. 0.0d0) then
            b(1) = 128
            b(2) = 0
            b(3) = 0
            b(4) = 0
            up = dummyx
            return
      endif
c
c
      b(4)=b(4)+1
      if (b(4) .ne. 0) then
            up = dummyx
            return
      endif
      b(3) = b(3) + 1
      if ( b(3) .ne. 0) then
            up = dummyx
            return
      endif
      b(2)= b(2)+1
      if ( b(2) .ne. 0) then
            up = dummyx
            return
      endif
      b(1)=b(1)-1
      if( ( sign(1.b(1)) .ne. sign(1,oldb1)) .or. ( b(1).eq.0) ) then
            write (*,*)'owerflow'
            stop
      endif
      up = dummyx
      return
      end
c
      real*8 function down(x)
```

```
      real*8 x,dummyx
      integer*2 b(4)
c
      equivalence( dummyx ,b)
      dummyx = x
c
      if ( dummyx/2.0d0 .eq. 0.0d0) then
            down = 0.0d0
            return
      endif
c
c     observe that we do not do any especial handling of zero.
c     We have constructed the other subroutines in such a way
c     that down has never to deal with zero
c
      b(4) = b(4)-1
      if ( b(4) .ne. -1) then
            down = dummyx
            return
      endif
      b(3)=b(3)-1
      if ( b(3) .ne. -1) then
            down = dummyx
            return
      endif
      b(2) = b(2)-1
      if ( b(2) .ne. -1) then
            down = dummyx
            return
      endif
      b(1) = b(1)-1
      down = dummyx
      return
c
c     notice that , since this was subroutine was running
c     over some non-zero number , it should find something on
c     the exponent. so no exception occurs; we can only end up
c     with zero but this is O.K.
c
      end
c
c
      real*8 function right(x)
      real*8 x
      real*8 up,down
```

```
      if ( x .ge. 0.0d0) then
              right = up(x)
      else
              right = down(x)
      endif
      return
      end
c
      real*8 function left(x)
      real*8 x
      real*8 right
      left = -right(-x)
      return
      end
```