Aperiodicity of the Hamiltonian Flow in the Thomas–Fermi Potential

Charles L. Fefferman*

Department of Mathematics, Princeton University

Luis A. Seco

Department of Mathematics, California Institute of Technology

"...que para sacar una verdad en limpio
menester son muchas pruebas y repruebas."

"Don Quijote de la Mancha", **M. de Cervantes.**

In [FS1] we announced a precise asymptotic formula for the ground–state energy of a non–relativistic atom. The purpose of this paper is to establish an elementary inequality that plays a crucial role in our proof of that formula. The inequality concerns the Thomas–Fermi potential $V_{TF}(r) = -y(ar)/r$, $a > 0$, where $y(r)$ is defined as the solution of

$$\left.\begin{aligned} y''(x) &= x^{-1/2} y^{3/2}(x) \\ y(0) &= 1 \\ y(\infty) &= 0 \end{aligned}\right\} \qquad (1.1)$$

(Without loss of generality, in what follows we will take $a = 1$.)
Define

$$F(\Omega) = F_y(\Omega) = \int \left(\frac{y(x)}{x} - \frac{\Omega^2}{x^2} \right)_+^{1/2} dx \qquad \Omega \in (0, \Omega_c)$$

* Partially supported by an NSF grant at Princeton University

where

$$\Omega_c^2 = \sup_{r>0} u(r) = u(r_c) \qquad u(r) = ry(r)$$

The subscript for F will be used whenever we want to emphasize the dependence of F on y.

Then, $F(\Omega)$ depends smoothly on Ω ([SW2]), and our main result here is as follows:

Theorem 1.1:

$$F''(\Omega) \leq c < 0 \qquad \textit{for all } \Omega \in (0, \Omega_c) \tag{1.2}$$

This is a quantitative form of the non–periodicity of almost all zero–energy orbits for the Hamiltonian

$$H = |\xi|^2 + V_{TF}(|x|)$$

on

$$\mathbf{R}^6 = \{(x, \xi) \mid x \in \mathbf{R}^3 \quad \xi \in \mathbf{R}^3\}$$

In fact, an easy computation shows that a zero–energy orbit with angular momentum Ω is periodic if and only if the derivative $F'(\Omega)$ is a rational multiple of π (see [Ar].) Hence, Theorem 1.1 shows that closed zero–energy orbits arise for only countably many Ω.

Theorem 1.1 will be used in our later papers ([FS5] and [FS6]) to control the density and eigenvalue sum arising from the three dimensional Schrödinger operator

$$H_Z = -\Delta + Z^{4/3} V_{TF}\left(Z^{1/3}|x|\right)$$

for large Z.

Aperiodicity of zero–energy Hamiltonian paths is well-known to play a crucial role in the study of eigenvalues and eigenfunctions. In our setting, Theorem 1.1 enters because our formulas for the eigenvalue sum and density involve expressions of the form

$$S = \sum_{1 \leq l < Z^{1/3}\Omega_c} \beta\left(\frac{Z^{1/3}}{\pi} F(Z^{-1/3} l)\right)$$

for elementary functions such as $\beta(t) = t - [t] - \frac{1}{2}$. (Here $[t]$ is the greatest integer in t.) Since β is bounded, we obtain trivially the estimate $S = O\left(Z^{1/3}\right)$. If $F(\Omega) = \pi\mu\Omega + \nu$

with μ rational, then the trivial estimate for S is easily seen to be the best possible. On the other hand, if $d^2F/d\Omega^2 < c < 0$, then one can prove that the numbers

$$\phi_l = Z^{1/3} F(Z^{-1/3} l)$$

are equidistributed modulo π. (The argument is close to Hardy's estimates on the number of lattice points in a disc.) Since $\beta(t)$ is periodic and has average zero, it follows that $S = O(Z^\gamma)$ with $\gamma < \frac{1}{3}$.
Thus, Theorem 1.1 allows us to improve on the trivial estimate for the sum S, which appears in the eigenvalue sum and density for H_Z. The complete proof of our results on atoms is contained in this paper together with [FS2], [FS3], [FS4], [FS5], [FS6] and [FS7].

The proof of Theorem 1.1 is necessarily rather delicate. For small perturbations of V_{TF} in a natural topology, the analog of Theorem 1.1 fails. Therefore, we have to make strong use of the differential equation defining $y(r)$. Our proof uses computer–assisted methods to solve that equation and to obtain bounds for F''. We remark, however, that without a computer it can also be seen that F'' vanishes at most finitely many times (Proposition 4.8 below; see also the recent independent proof in [HKSW]), which also implies that zero–energy periodic orbits have measure zero, which in turn also implies the same results stated above for sums S, and therefore our result for atomic energies. Theorem 1.1, however, is better because it implies better error terms for all those formulas. Moreover, if one wants to understand ground–state energies to a greater accuracy, then Theorem 1.1, with all its strength, is unavoidable.

In what follows, our proofs will *not* be computer–assisted unless stated otherwise.

It would be interesting to prove the aperiodicity of almost all zero–energy Hamiltonian paths in the Thomas–Fermi potential for a molecule.

The complete programs used in our proof are publicly available by anonymous `ftp` from the machine `math.utexas.edu` (Internet number 128.83.133.215) This machine also supports other standard methods of such as `gopher` and `wais`. The interested parties should contact their administrators about availability and usage of these programs on their machine. The machine `math.utexas.edu` has a user called `anonymous` whose password is the e-mail address of the actual user. Our programs are stored in the directory `/pub/papers/feffsec`. We refer the reader to the file `README` there for instructions on how to download the programs. Each one of them has instructions on how to use them.

More information about how to interact with `math.utexas.edu` is available from the Mathematical Physics Preprint Archive. In particular, the user can obtain detailed instructions on how to install the public domain programs `gopher` and `wais`. Send e-mail to `mp_arc@math.utexas.edu` for details.

We also remark that the American Mathematical Society maintains the `e-math` account in the machine `e-math.ams.com` (Internet number 130.44.1.100). This account includes a menu, one of whose entries is gopher. At the moment, the mp_arc gopher connection is in the main menu. Going through different submenus, one can also reach the U.T. Math. gopher server. The user may find out other machines that provide public access to Internet services.

1. Preliminaries.

In this section we consider a smooth function y that looks like the Thomas–Fermi function. More precisely, let $u(x) = xy(x)$; then, we assume the following holds;

a. $y > 0$, $y(0) = 1$ and $\lim_{x\to\infty} y(x) = 0$.

b. There exists a point r_c s.t. $u(x) < u(r_c)$ for $x \neq r_c$, $u'(x) > 0$ for $0 \leq x \leq r_c$, and $u'(x) < 0$ for $r_c \leq x$. Also, $u''(r_c) < 0$.

We will denote the two solutions of $u(r) = \Omega^2$ by $r_1(\Omega) < r_2(\Omega)$. We start by giving convenient formulas for the derivatives of F. We point out that similar formulas were given in [SW2]. One of the reasons we need formulas of the kind stated below is to obtain expressions such as (1.7) and (1.8) below. Also, we will see that in the case of an analytic y, not only is F analytic on $(0, \Omega_c)$, but it admits an analytic extension beyond Ω_c. However, 0 will be in general an essential singularity.

Lemma 1.2: *Let y be as above. The following formulas hold:*

$$F(\Omega) = \int \left(u(x) - \Omega^2\right)_+^{1/2} \frac{dx}{x}$$

$$F'(\Omega) = -\Omega \int \left(u(x) - \Omega^2\right)_+^{-1/2} \frac{dx}{x}$$

$$F''(\Omega) = -\lim_{\delta \to 0} \left(\int_{r_1(\Omega)+\delta}^{r_2(\Omega)-\delta} \left(u(x) - \Omega^2\right)^{-3/2} y(x)\, dx + c(\Omega)\delta^{-1/2} \right)$$

where $c(\Omega)$ is uniquely specified by requiring the finiteness of the limit. Moreover, if b is any number less than $r_2(\Omega)$, then

$$\frac{d^2}{d\Omega^2} \int_{r_1(\Omega)}^{b} \left(u(x) - \Omega^2\right)_+^{1/2} \frac{dx}{x}$$

equals

$$-\lim_{\delta \to 0} \left(\int_{r_1(\Omega)+\delta}^{b} \left(u(x) - \Omega^2\right)^{-3/2} y(x)\, dx + c_1(\Omega)\delta^{-1/2} \right)$$

again, for a constant c_1 that makes the limit finite. The corresponding symmetric case also holds.

Proof: The first two formulas are trivial. For the third, let

$$H(\delta, \Omega) = \Omega \int_{r_1(\Omega)+\delta}^{r_2(\Omega)-\delta} \left(u(r) - \Omega^2\right)^{-1/2} \frac{dr}{r}$$

Note that the formula for F'' amounts to showing that

$$\frac{d}{d\Omega} \lim_{\delta \to 0} H(\delta, \Omega) = \lim_{\delta \to 0} \frac{d}{d\Omega} H(\delta, \Omega) \tag{1.3}$$

Indeed, the left hand side equals $-F''$, whereas the right hand side equals

$$\lim_{\delta \to 0} \Bigg\{ \Omega^2 \int_{r_1(\Omega)+\delta}^{r_2(\Omega)-\delta} \left(u(r) - \Omega^2\right)^{-3/2} \frac{dr}{r} + \int_{r_1(\Omega)+\delta}^{r_2(\Omega)-\delta} \left(u(r) - \Omega^2\right)^{-1/2} \frac{dr}{r}$$
$$+\Omega \Big(\frac{\left(u\left(r_2 - \delta\right) - u(r_2)\right)^{-1/2}}{r_2 - \delta} r_2'(\Omega) - \frac{\left(u\left(r_1 + \delta\right) - u(r_1)\right)^{-1/2}}{r_1 + \delta} r_1'(\Omega) \Big) \Bigg\}$$
$$= \lim_{\delta \to 0} \Bigg\{ \int_{r_1(\Omega)+\delta}^{r_2(\Omega)-\delta} \left(u(r) - \Omega^2\right)^{-3/2} u(r) \frac{dr}{r}$$
$$-\Omega \sum_{i=1,2} \frac{|u'(r_i)|^{-1/2} |r_i'(\Omega)| \delta^{-1/2} (1 + O(\delta))}{r_i(\Omega)} \Bigg\}$$

which agrees with the formula asserted for $-F''$, provided that this previous expression for $c(\Omega)$

$$c(\Omega) = -\Omega \sum_{i=1,2} \frac{|u'(r_i)|^{-1/2} |r_i'(\Omega)|}{r_i(\Omega)}$$

actually makes the limit above finite.

Therefore, the lemma will follow if we show that both $H(\delta,\Omega)$ and $\frac{\partial}{\partial\Omega}H(\delta,\Omega)$ converge uniformly on compact subsets of $(0,\Omega_c)$ to C^1 functions. This will imply, first, that we can interchange limits in (1.3), and, second, that the expression for $c(\Omega)$ above is the right one.

In order to see this, consider the change of variables given by

$$t(r) = \begin{cases} \left(\Omega_c^2 - u(r)\right)^{1/2} & \text{if } r \geq r_c \\ -\left(\Omega_c^2 - u(r)\right)^{1/2} & \text{if } r \leq r_c \end{cases} \tag{1.4}$$

Note that t is smooth and strictly increasing in the range $(0,\infty)$. We can therefore consider its inverse, $r(t)$, and use it to rewrite

$$H(\delta,\Omega) = \Omega \int_{t_1(\delta,\Omega)}^{t_2(\delta,\Omega)} \left(D^2 - t^2\right)^{-1/2} w(t)\,dt$$

where

$$t_1 = t(r_1+\delta) \qquad t_2 = t(r_2-\delta) \qquad D^2 = \Omega_c^2 - \Omega^2 \qquad w(t) = \frac{r'(t)}{r(t)}$$

Note that w is smooth on $(0,\Omega_c)$, and that

$$t_1 = -D\left(1+\tau_1(\delta)\right) \qquad t_2 = D\left(1+\tau_2\left(\delta\right)\right) \qquad c\delta \leq |\tau_i| \leq C\delta \quad \text{for } i=1,2 \tag{1.5}$$

uniformly on compact subsets of $(0,\Omega_c)$, which implies that

$$H(\delta,\Omega) = \Omega \int_{D^{-1}t_1}^{D^{-1}t_2} \left(1-t^2\right)^{-1/2} w(tD)\,dt$$

converges uniformly to the C^1 function

$$H(0,\Omega) = \Omega \int_{-1}^{1} \left(1-t^2\right)^{-1/2} w(tD)\,dt = -F'(\Omega). \tag{1.6}$$

As for $\frac{d}{d\Omega}H(\delta,\Omega)$,

$$\frac{d}{d\Omega}H(\delta,\Omega) = \int_{D^{-1}t_1}^{D^{-1}t_2} (1-t^2)^{-1/2} \frac{\partial}{\partial\Omega}\Big(\Omega w(tD)\Big)\, dt + \Omega \sum_{i=1,2} G_i(\delta,\Omega)$$

with

$$G_i(\delta,\Omega) = \pm\left(1 - D^{-2}t_i^2\right)^{-1/2} w(t_i) \frac{\partial}{\partial\Omega}\left(D^{-1}t_i\right)$$

The first term above converges with δ to the smooth function

$$\int_{-1}^{1} (1-t^2)^{-1/2} \frac{\partial}{\partial\Omega}\Big(\Omega w(tD)\Big)\, dt$$

uniformly on compact subsets of $(0,\Omega_c)$. Thus, the lemma will follow if we prove that G_i goes to zero with δ uniformly in Ω. By (1.5), this will in turn follow if we prove that

$$\frac{\partial}{\partial\Omega}\left(D^{-1}t_i\right) = O(\delta)$$

By (1.5) again, it is enough to prove that

$$\frac{\partial}{\partial\Omega}\left(D^{-1}t_i\right)^2 = O(\delta)$$

But, for $i=1$,

$$\begin{aligned}
\frac{\partial}{\partial\Omega}\left(D^{-1}t_1\right)^2 &= \frac{\partial}{\partial\Omega}\left(\frac{u(r_1+\delta)-\Omega_c^2}{u(r_1)-\Omega_c^2}\right) \\
&= \frac{\left(u(r_1)-\Omega_c^2\right)u'(r_1+\delta)r_1'(\Omega) - \left(u(r_1+\delta)-\Omega_c^2\right)u'(r_1)r_1'(\Omega)}{\left(u(r_1)-\Omega_c^2\right)^2} \\
&= \frac{r_1'(\Omega)}{\left(u(r_1)-\Omega_c^2\right)^2}\cdot \\
&\qquad \cdot\Big(\left(u(r_1)u'(r_1+\delta) - u(r_1+\delta)u'(r_1)\right) - \Omega_c^2\left(u'(r_1+\delta)-u'(r_1)\right)\Big)
\end{aligned}$$

The first factor above is trivial. The other is clearly bounded by $C\delta$, and, doing the same for $i=2$, the lemma follows.

The last remark in the statement of the lemma follows in exactly the same way, with the only modification that one of the G_i is in fact constant in δ, which of course does not affect the uniform approach to a C^1 function. QED

A closer look at (1.6) yields the following remark:

Corollary 1.3: *Define $w(t)$ as in the proof of the previous lemma. Then*

$$-F''(\Omega) = \int_{-1}^{1} (1-t^2)^{-1/2} \frac{\partial}{\partial \Omega}\Big(\Omega w(tD)\Big)\, dt$$

In particular, if $y \in C^k(0,\infty)$, then $F_y \in C^{k-1}(0,\Omega_c)$, $k \geq 2$. Also, if y is analytic $F(\Omega)$ admits an analytic extension to a complex neighborhood of $(0,\Omega_c]$.

Proof: If $y \in C^k$, the same is true for u. Therefore, $t \in C^{k-1}$, thus $r \in C^{k-1}(-\Omega_c,\Omega_c)$ and $r(t) \neq 0$, which implies $w \in C^{k-2}(-\Omega_c,\Omega_c)$, and, by (1.6), $F' \in C^{k-2}$.

In the case of an analytic y, since w is analytic in some neighborhood around 0, it admits a convergent power series expansion

$$w(t) = \sum_{n=0}^{\infty} w_n t^n \qquad t < t_0 \tag{1.7}$$

This implies

$$-F'(\Omega) = \Omega \sum_{n=0}^{\infty} w_{2n} D^{2n} \int_{-1}^{1} (1-t^2)^{-1/2} t^{2n}\, dt \tag{1.8}$$

since the odd terms clearly yield an integral 0, and thus drop out of the sum. This, in particular shows that F can be defined as an analytic function around Ω_c. Since, by (1.6), F is analytic also in $(0,\Omega_c)$, the corollary follows. QED

We will see later (Proposition 4.8) that the limit

$$\lim_{\Omega \to 0} F''(\Omega)\Omega^{\gamma} \qquad \gamma = \frac{9-\sqrt{73}}{2} > 0$$

exists, is finite and not zero. This shows, in particular, that F has an essential singularity at 0 and that F is not a linear function.

The proof of (1.2) will now go as follows:
We make an initial division of $(0,\Omega_c)$ into two intervals $(0,\bar{\Omega})$ and $[\bar{\Omega},\Omega_c]$, that we will refer to as Zone I and Zone II, respectively. In Zone I, we will use the formula in

Lemma 1.2 to prove (1.2) uniformly on very little subintervals of $(0, \bar{\Omega})$. We will deal with this in Section 4.
Then, formula (1.8) will allow us to show (1.2) uniformly on Zone II, as explained in Section 5.

Our proof will rely on a very precise knowledge of the solution to the Thomas–Fermi equation. For this, we will use computer assisted techniques. The next section deals with a description of how the computer will be used to yield theorems.

Acknowledgments We wish to express our deepest gratitude to R. de la Llave: in addition to stimulating conversations, he taught us everything we know about computer–assisted proofs, gave us useful advice concerning the presentation of the paper, and went through the excruciating pain of checking our computer programs. We are also grateful to D. Rana for providing us with his interval arithmetic package. Finally, we thank the Department of Mathematics of the University of Texas at Austin for their help with the electronic distribution of the computer programs.

2. Computer–Assisted Analysis

Let $\mathcal{R}$ be the set of "representable numbers" in a computer, that is those numbers that the computer can represent exactly. Depending on the specific machine, they are usually real numbers with some finite binary expansion.

It is well known that computers can only perform arithmetic in an approximate way: the addition —for example— of two representable numbers is another representable number that will probably be close to the true sum, but is not exactly the true sum.

The idea to perform rigorous arithmetic is to instruct the computer on how to produce upper and lower bounds to the true results of arithmetic operations between representable numbers; in other words, we work with intervals with endpoints in $\mathcal{R}$, and we implement arithmetic operations on intervals in such a way that given two intervals, the computer will produce a third that is guaranteed to contain the result of all arithmetic operations between points in the initial intervals. This is usually called "interval arithmetic".

We denote the set of all these intervals by $\mathcal{I}$. Also, given a real function $f(x)$, we denote

$$f(I) = \{f(x) \mid x \in I\} \qquad I \in \mathcal{I}$$

Binary functions of intervals are defined accordingly. In particular, a statement like $I_1 > I_2$ means that $x > y$ for all pairs (x, y), $x \in I_1$, $y \in I_2$. Also, given $I = [a, b]$ and $\epsilon \geq 0$, we introduce the shorthand notation $I \pm \epsilon$ to denote an interval containing $[a - \epsilon, b + \epsilon]$. We also point out, although it really is redundant, that in what follows, finite decimal expressions for numbers represent the rational numbers with exactly those decimal expansions.

The next step is to perform a similar kind of arithmetic, but where objects are functions in some Banach space, not numbers. A convenient Banach space to use in this theory is the space of piecewise analytic functions, with a lower bound on the size of the domains of analyticity.

Occasionally, it will be convenient to switch to genuine real variable theory, for which we will do our work on $C^0[-1,1]$. The reason for this is that inversion of functions in $\mathbf{R}^1$ is a little easier than the complex counterpart, mainly because the domain of definition problem is trivial in the real case. We remark though, that the use of C^0 is not essential, and the same analysis could be carried over to H^1 with a little more work.

More precisely, consider the Banach Algebras

$$H^1 = \left\{ f(z) \;\mid\; f(z) = \sum_{n=0}^{\infty} a_n\, z^n, \quad \sum_{n=0}^{\infty} |a_n| < \infty \right\}$$

and

$$C^0 = \{f(x) \;\mid\; f \text{ is continuous on } [-1,1]\}$$

with norms

$$\|f\|_1 = \sum_{n=0}^{\infty} |a_n| \qquad \|f\|_\infty = \sup |f(x)|$$

respectively.

H^1 is a subspace of the set of analytic functions in the unit disk.

Then, our substitute for intervals are sets $\mathcal{U}^1(I_0, \ldots, I_N; C_h, C_g; k)$ of the form

$$\left\{ f(z) = \sum_{n=0}^{\infty} a_n \, z^n + z^k g(z) \, \middle| \, a_n \in I_n, \quad 0 \le n \le N, \sum_{n=N+1}^{\infty} |a_n| \le C_h \, , \, \|g\|_1 \le C_g \right\} \tag{2.1}$$

where C_h and C_g are positive real numbers and I_n are intervals in the real line. The parameter k will generally be problem–dependent and fixed. For the computer implementation, C_h and C_g will run over the set of computer–representable numbers, and the intervals will be those with representable endpoints. We refer to C_h and C_g as high and general order error terms respectively, for obvious reasons. If intervals have nonempty interior and $C_h > 0$, or if $k = 0$ and $C_g > 0$, then these sets are in fact a neighborhood basis for the topology induced by $\| \, \|_1$. For this reason, we will refer to these $\mathcal{U}$ as "neighborhoods", even if in general they will not be. We will refer to them as neighborhoods of **type** k whenever we want to emphasize the integer k in definition (2.1). If $C_g = 0$, we refer to them as type ∞. In general, $\mathcal{U}(k)$ means that $\mathcal{U}$ is a neighborhood of type k. Also, we will refer to them as being of **order** N to indicate that they consist of $N + 1$ intervals. In our implementation, N will not be fixed, but chosen adaptatively during the execution of the programs.

The reason why this is a convenient space to work in is because elementary operations, such as addition, product, integration, differentiation (composed with a slightly contracting dilation), evaluation at a point and integration of initial value problems in ordinary differential equations can be conveniently bounded by elementary formulas in terms of this set of neighborhoods.
By trivial scaling, we will be able to do analysis on

$$H^1(|z - z_0| \le r) = \left\{ f(z) \;\middle|\; f(z) = \sum_{n=0}^{\infty} a_n \left(\frac{z - z_0}{r} \right)^n , \quad \sum_{n=0}^{\infty} |a_n| < \infty \right\}$$

a subspace of the set of analytic functions on the disk of center z_0 and radius r.

As for C^0, we will use sets (that we will also refer to as "neighborhoods") of the type:

$$\begin{aligned} \mathcal{U}^0(I_0, \ldots, I_N; C_h, C_g; k; S) = \Bigg\{ f(z) = \sum_{n=0}^{N} a_n \, z^n + z^{N+1} h(z) + z^k g(z) \, \Bigg| \\ a_n \in I_n, \quad 0 \le n \le N, \qquad \sup_{z \in S} |h(z)| \le C_h \, , \, \sup_{z \in S} |g(z)| \le C_g \Bigg\} \end{aligned} \tag{2.2}$$

where S is a subset of $[-1,1]$, and h and g are continuous functions on S.
We will use the superscript 0 or 1 whenever we want to emphasize in which topology we are taking these "neighborhoods".
Note the natural inclusion

$$\mathcal{U}^1(I_0,\ldots,I_N;C_h,C_g;k) \subset \mathcal{U}^0(I_0,\ldots,I_N;C_h,C_g;k;S)$$

for any $S \subset [-1,1]$.
These sets of neighborhoods $\mathcal{U}^0(k)$ will not allow us to perform as many operations as their smaller brothers the $\mathcal{U}^1(k)$, but we can still add, multiply, raise to fractional powers and integrate (among others) in terms of them; furthermore, the formulas for these neighborhood operations are exactly the same as those for the $\mathcal{U}^1(k)$.

We illustrate this neighborhood analysis describing how we can raise neighborhoods to real powers. At this point, we make the following remark concerning our use and description of algorithms:
Algorithms describe a procedure that, if successful, will allow us to construct (usually upper and lower bounds for) certain numbers. When we describe these algorithms, we will state under which conditions they *fail*; a failure means that the procedure is stopped, an error reported, and no theorem proved. Obviously, if during the description of an algorithm, we use another algorithm, a failure in the execution of the latter implies also a failure of the former algorithm.

Lemma 2.1: *Let $0 < r < 1$. Then*

$$\sup_{n \geq N} (nr^n) \leq \begin{cases} Nr^N & \text{if } N \geq \frac{1}{|\log r|} \\ \frac{1}{e|\log r|} & \text{otherwise} \end{cases}$$

Proof: The function xr^x attains its maximum when $x = |\log r|^{-1}$. QED

Lemma 2.2: *Consider, in any commutative Banach Algebra, the operators*

$$T^\alpha(y) = (1+y)^\alpha$$

acting on $\|y\| \le r < 1$. *Then, we have*

$$\|T^\alpha(y)\| \le K_{2.2}(\alpha, \|y\|)$$
$$\|T^\alpha\|_{\mathrm{Lip}} \le C_{2.2}(\alpha, r)$$

where

$$K_{2.2}(\alpha, \|y\|) \quad \stackrel{\text{def}}{\equiv} \quad \begin{cases} \min\left((1-\|y\|)^{-|\alpha|}\,,\ 1+|\alpha|\dfrac{\|y\|}{1-\|y\|}\right) & -1 \le \alpha \le 2 \\ \\ (1-\|y\|)^{-|\alpha|} & \textit{otherwise} \end{cases}$$

$$C_{2.2}(\alpha, r) \quad \stackrel{\text{def}}{\equiv} \quad \begin{cases} \min\left(|\alpha| + r\dfrac{|\alpha|\,|\alpha-1|}{(1-r)^2}\,,\ |\alpha|(1-r)^{-|\alpha-1|}\right) & -1 \le \alpha \le 2 \\ \\ |\alpha|(1-r)^{-|\alpha-1|} & \textit{otherwise} \end{cases}$$

Proof: First, if $-1 \le \alpha \le 2$ and $\|y_1\|$, $\|y_2\| \le r$,

$$(1+y_1)^\alpha - (1+y_2)^\alpha = \sum_{n=1}^{\infty} \binom{\alpha}{n} (y_1^n - y_2^n)$$

Now,

$$y_1^n - y_2^n = (y_1 - y_2) \sum_{k=0}^{n-1} y_1^k y_2^{n-1-k}$$

and

$$\left\| \sum_{k=0}^{n-1} y_1^k y_2^{n-1-k} \right\| \le n r^{n-1}$$

Since $2 \ge \alpha \ge -1$, $\left|\binom{\alpha}{n}\right|$ is a decreasing sequence in n, for $n \ge 1$. Therefore,

$$\begin{aligned} \frac{\|T(y_1) - T(y_2)\|}{\|y_1 - y_2\|} &\le \sum_{n=1}^{\infty} \left|\binom{\alpha}{n}\right| n r^{n-1} \\ &\le |\alpha| + r\frac{|\alpha|\,|\alpha-1|}{(1-r)^2} \end{aligned}$$

On the other hand, for α in the same range,

$$\begin{aligned} \|T(y)\| &\le \sum_{n=0}^{\infty} \left|\binom{\alpha}{n}\right| \|y\|^n \\ &\le 1 + |\alpha|\,\frac{\|y\|}{1-\|y\|} \end{aligned}$$

Now, for general α, and $\|y_1\|, \|y_2\| \le r$, note that

$$\|e^y\| \le e^{\|y\|}$$

and

$$\|\log(1+y)\| \le -\log(1-\|y\|)$$

Therefore,

$$\|T^\alpha(y)\| = \left\|e^{\alpha \log(1+y)}\right\| \le (1-\|y\|)^{-|\alpha|}$$

and

$$\begin{aligned}\|T^\alpha(y_1) - T^\alpha(y_2)\| &= |\alpha| \left\|(y_1 - y_2)\int_0^1 T^{\alpha-1}\big(ty_1 + (1-t)y_2\big)\,dt\right\| \\ &\le |\alpha|\,\|y_1 - y_2\|\,(1-r)^{-|\alpha-1|}\end{aligned}$$

QED

Algorithm 2.3: *Given a neighborhood $\mathcal{U}(I_0, \ldots, I_N; C_h, C_g; k)$ satisfying*

1. $I_0 > 0$.
2. $|I_0| > \sum_{n>0} |I_n| + C_h + C_g$.
3. *if* $\alpha > 2$, *then* $2N > \alpha - 1$.

we construct another, $\tilde{\mathcal{U}}(k)$, such that, if $f \in \mathcal{U}$ then $f^\alpha \in \tilde{\mathcal{U}}(k)$.
The algorithm is independent of k, and of whether the neighborhoods are in H^1 or C^0.
If $C_g = 0$ for $\mathcal{U}$, then the same is true for $\tilde{\mathcal{U}}$.

Description: Assume first that $\alpha \ge -1$.
Let $f \in \mathcal{U}$. Put $\tilde f = \big(f(0)\big)^{-1} \cdot f$, so $\tilde f = 1 + y(z) + z^k g(z)$, where $y(z) = z\tilde y(z)$,

$$1 + y(z) \in \mathcal{U}(I'_0, \ldots, I'_N; C'_h, 0; 0) \tag{2.3}$$

for $I'_i = f(0)^{-1} \cdot I_i$, $C'_h = f(0)^{-1} \cdot C_h$, and $\|g\| \le C_g/f(0)$. Bounds for all this can be computed easily since we know that $f(0) \in I_0$.
Now,

$$\big(1 + y(z)\big)^\alpha = \sum_{n=0}^N \binom{\alpha}{n} y(z)^n + h(z)$$

where $h(z) = z^{N+1}\tilde{h}(z)$. In the H^1 topology, $\|h\|_1 = \|\tilde{h}\|_1$, and

$$\begin{aligned}\|\tilde{h}\|_1 &\le \sum_{n>N} \left|\binom{\alpha}{n}\right| \|y\|_1^n \\ &\le \left|\binom{\alpha}{N+1}\right| \frac{r^{N+1}}{1-r}\end{aligned}$$

for

$$r = \sum_{i=1}^{N} |I_i'| + C_h'$$

where we have used condition 3. in the statement of the algorithm. In the C^0 topology,

$$\begin{aligned}\left|\tilde{h}(z)\right| &\le \sum_{n>N} \left|\binom{\alpha}{n}\right| \left|\frac{y(z)}{z}\right|^n \\ &\le \left|\binom{\alpha}{N+1}\right| \frac{r^{N+1}}{1-r}\end{aligned}$$

for

$$r = \sum_{i=1}^{N} |I_i'| + C_h'.$$

As a result of this, the computation of $\|\tilde{h}\|$ is done in exactly the same way whether we are in the C^0 or H^1 topologies.

Concerning the computation of the factor $\frac{1}{1-r}$, it is done as follows: we first check that $r \in (0,1)$ the check for $r > 0$ being unnecessary, harmless but convenient; then, we compute an upper bound for $\frac{1}{1-r}$ with our interval arithmetic package, knowing that an overflow will be reported and the program terminated if we cannot find such upper bound with machine–numbers.

Also,

$$(1 + y(z))^\alpha - \left(\tilde{f}(z)\right)^\alpha = O\left(z^k\right)$$

which implies that general errors are of type k. In the case that we are in H^1, since multiplication by z is an isomorphism, by Lemma 2.2, we see that general errors are bounded by

$$\left\|(1 + y(z))^\alpha - \tilde{f}(z)^\alpha\right\| \le \|g\| \, C_{2.2}(\alpha, \|y\| + \|g\|)$$

If, however, we are in C^0, apply Lemma 2.2 to ($\mathbf{R}^1$,+,$\cdot$), to get

$$\begin{aligned}\left|(1 + y(z))^\alpha - \tilde{f}(z)^\alpha\right| &\le \left|1 + y(z) - \tilde{f}(z)\right| \cdot C_{2.2}\left(\alpha, \|g\|_\infty + \|y\|_\infty\right) \\ &\le |z^k| \cdot \|g\|_\infty \cdot C_{2.2}\left(\alpha, \|g\|_\infty + \|y\|_\infty\right)\end{aligned}$$

since $|y(z)|$, $|\tilde{f}(z) - 1| \le \|g\|_\infty + \|y\|_\infty$ and $C_{2.2}(\alpha, t)$ is increasing in t.

Therefore, say that

$$\sum_{n=0}^{N} \binom{\alpha}{n} y(z)^n \in \mathcal{U}_1(\tilde{I}_0, \ldots, \tilde{I}_N; \tilde{C}_h, 0; \infty)$$

by (2.3). Then,

$$\left(\tilde{f}\right)^{\alpha} \in \mathcal{U}(\tilde{I}_0, \ldots, \tilde{I}_N; \tilde{\tilde{C}}_h, \tilde{\tilde{C}}_g; k)$$

with

$$\tilde{\tilde{C}}_h = \tilde{C}_h + \left| \binom{\alpha}{N+1} \right| \frac{r^{N+1}}{1-r}$$

and

$$\tilde{\tilde{C}}_g = f(0)^{-1} C_g \cdot C_{2.2}\left(\alpha, \|y\| + C_g f(0)^{-1}\right)$$

and

$$f^{\alpha} \in f(0)^{\alpha} \cdot \mathcal{U}(\tilde{I}_0, \ldots, \tilde{I}_N; \tilde{\tilde{C}}_h, \tilde{\tilde{C}}_g; k)$$

In the case $\alpha < -1$, we can find an integer k such that $2^{-k}\alpha \ge -1$. Then, we can find a neighborhood containing $f^{2^{-k}\alpha}$. By ordinary multiplication we can thus construct a neighborhood containing $f^{\alpha} = \left(f^{2^{-k}\alpha}\right)^{2^k}$. QED

Although computer–assisted analysis has become fairly standard, we refer the reader to [Mo] and [KM] for a description of the basic ideas. The technique for solving ODE's is adapted from [Se2] and [Se1], and is tailored to handle our particular ODE. See [Lo] for a thorough discussion on ODE solving techniques, with very good general algorithms. Also, we refer the reader to [EKW], [EW], [FL], [LL], [Ll] and [Ra] for a sample of computer–assisted proofs of a wide variety of problems. Main ideas in our approach go back to those proofs.

Our interval arithmetic package is an adaptation of the one used in [Se1] and [Se2], which in turn is an adaptation of the one developed by D. Rana. See [Ra] and [Se1] for details on the software.

3. The Thomas–Fermi Equation

In this section we will be concerned with the problem of getting good bounds for the solution of the Thomas–Fermi equation (1.1).
It is well known ([Hi]) that

$$-w_0 = \lim_{r\to 0} y'(r) < 0 \tag{3.1}$$

exists, and that y admits a power series expansion

$$y(r) = 144r^{-3}\Big(\sum_{n=0}^{\infty} b_n r^{-n\alpha}\Big) \tag{3.2}$$

convergent for r large enough, with $b_0 = 1$, $b_1 < 0$ and $\alpha = \frac{1}{2}(\sqrt{73} - 7)$.

Also, y is always positive, decreasing, and it is the only such solution of the ODE satisfying (3.1) and (3.2).

The Initial Value Problem away from the Singularities.
In this section we will be concerned with the solution to the Initial Value Problem

$$\left.\begin{aligned} u''(x) &= x^{-1/2} u^{3/2}(x) \\ u(x_0) &= u_0 \\ u'(x_0) &= u_1 \end{aligned}\right\} \tag{3.3}$$

for $x_0, u_0 > 0$
The solution to this problem will be in terms of a function $f \in H^1$ satisfying

$$u(x) = u_0 + u_1 \cdot r \cdot z + z^2 f(z)$$

where $z = (x - x_0)/r$ and r is a small positive representable number (in particular, $r < x_0$).
Note that the solution of (3.3) can be viewed as the fixed point of

$$T(u) = u_0 + \int_{x_0}^{x} \left(u_1 + \int_{x_0}^{t} \frac{u^{3/2}(s)}{s^{1/2}}\, ds \right) dt$$

and that T induces in a trivial way an operator $\tilde{T}$ of which f is its fixed point.

Throughout this section, we will do our work on H^1, and $\|\ \|$ will always denote $\|\ \|_1$.

Algorithm 3.1: *We deduce conditions on u_0, u_1, x_0, r and α under which $\tilde{T}$ is a well–defined contraction in $B(0,\alpha) \subset H^1$, and we compute an upper bound for $\left\|\tilde{T}\right\|_{\text{Lip}}$.*

Description: Let $g = \sum a_n z^n$. Consider the operators

$$\begin{aligned} T_1(f) &= r\, u_1\, z + z^2 f(z) \\ T_2(g) &= (u_0 + g(z))^{3/2} \\ T_3(g) &= (rz + x_0)^{-1/2} \cdot g \\ T_4(g) &= r^2 \sum_{n \geq 0} \frac{a_n z^n}{(n+1)(n+2)} \end{aligned} \tag{3.4}$$

It is clear that

$$T(u) = u_0 + r u_1 z + z^2 \,(T_4 \circ T_3 \circ T_2 \circ T_1)(f) \tag{3.5}$$

and thus, $\tilde{T} = T_4 \circ T_3 \circ T_2 \circ T_1$.
Now, T_1 is affine with an isometry as the linear part, and

$$\|T_4\|_{\text{Lip}} \leq \tfrac{1}{2} r^2 \tag{3.6}$$

Using Lemma 2.2, and putting $\beta = r/x_0$, we can see that

$$\begin{aligned} \|T_3\|_{\text{Lip}} &\leq \left\|(rz + x_0)^{-1/2}\right\|_1 \\ &\leq x_0^{-1/2} K_{2.2}(-\tfrac{1}{2}, \beta) \end{aligned} \tag{3.7}$$

Here we assume $\beta < 1$, otherwise we say the algorithm fails.
For T_2, we have

$$\|T_2\|_{\text{Lip}} \leq u_0^{1/2} C_{2.2}(\tfrac{3}{2}, \gamma) \tag{3.8}$$

whenever

$$\gamma \geq u_0^{-1} \sup_{\|f\| \leq \alpha} \|T_1(f)\|$$

Since

$$u_0^{-1} \sup_{\|f\| \leq \alpha} \|T_1(f)\| \leq \frac{r|u_1| + \alpha}{u_0} \quad \stackrel{\text{def}}{\equiv} \quad \gamma_0$$

we have

$$\left\|\tilde{T}\right\|_{\text{Lip}} \leq \tfrac{1}{2} r^2 x_0^{-1/2} u_0^{1/2} K_{2.2}(-\tfrac{1}{2}, \beta) C_{2.2}(\tfrac{3}{2}, \gamma_0) \tag{3.9}$$

Also, here we assume $\gamma_0 < 1$, or else the algorithm fails.

Next, we need to show that $\tilde{T}$ maps $B(0,\alpha)$ into itself. In order to do this, note that $\|T_4(g)\| \leq \frac{1}{2}r^2\|g\|$, which implies

$$\begin{aligned}\left\|\tilde{T}(0)\right\| &\leq \tfrac{1}{2}r^2\left\|(rz+x_0)^{-1/2}\right\| \cdot \left\|(u_0+ru_1z)^{3/2}\right\| \\ &\leq \tfrac{1}{2}r^2x_0^{-1/2}u_0^{3/2}K_{2.2}(-\tfrac{1}{2},\beta)K_{2.2}\left(\tfrac{3}{2},\frac{r|u_1|}{u_0}\right)\end{aligned}$$

Note that our assumption on γ_0 guarantees that the last term above is well–defined. Then, since

$$\left\|\tilde{T}(f)\right\| \leq \left\|\tilde{T}(0)\right\| + \alpha\left\|\tilde{T}\right\|_{\mathrm{Lip}}$$

we see that $\tilde{T}$ maps $B(0,\alpha)$ into itself provided

$$\tfrac{1}{2}r^2x_0^{-1/2}u_0^{3/2}K_{2.2}(-\tfrac{1}{2},\beta)K_{2.2}\left(\tfrac{3}{2},\frac{r|u_1|}{u_0}\right) \leq \alpha\,(1-L)$$

whenever L is an upper bound for $\left\|\tilde{T}\right\|_{\mathrm{Lip}}$. The algorithm also reports a failure if the upper bound L obtained using (3.9) is not strictly less than 1. QED

Note that if the previous conditions are satisfied, we also know that the solution u is strictly positive on $[x_0-r, x_0+r]$. Also, we know that it is defined as an analytic function on $|z-x_0|<r$.

Algorithm 3.2: *Given intervals x^*, u_0^* and u_1^*, and representable r, we construct a neighborhood $\mathcal{U}(I_0,\ldots,I_N;0,C_g;0)$ such that for any $x_0 \in x^*$, $u_0 \in u_0^*$, and $u_1 \in u_1^*$, and any solution u of (3.3) with any of these initial conditions, we have*

$$u(x) = u_0 + u_1\cdot(x-x_0) + z^2f(z) \qquad z = \frac{x-x_0}{r}$$

for some $f \in \mathcal{U}$.
We can also make that neighborhood to have the form $\mathcal{U}(I_0,\ldots,I_N;C_h,0;\infty)$.

Description: First, we construct, in a heuristic way, a polynomial

$$p(z) = \sum_0^N p_i z^i$$

which approximately solves $\tilde{T}p = p$, and we set α such that $\|p\| \leq \alpha$. Next, we look for $\alpha_0 \geq \alpha$ such that the conditions on x_0, u_0, u_1, r and α_0 given by Algorithm 3.1 hold uniformly for all $x_0 \in x^*$, $u_0 \in u_0^*$ and $u_1 \in u_1^*$.
Next, since f is the fixed point of $\tilde{T}$, we have

$$\|p - f\| \leq \frac{\left\|p - \tilde{T}p\right\|}{1 - \left\|\tilde{T}\right\|_{\mathrm{Lip}}}$$

Now, formulas (3.4) and (3.5) allow us to compute an upper bound for the numerator, Algorithm 3.1 allows us to compute a lower bound for the denominator, and we set C_g to be the resulting upper bound for the ratio. This immediately yields the required $\mathcal{U}$, by putting $I_i = [\,p_i\,, p_i\,]$ for $i = 0, \ldots, N$.

In order to obtain neighborhoods of type ∞, note that by power matching, for a given i, we can produce an interval I_i that contains any of the i'th Taylor coefficient for any of the solutions to the ODE for all $x_0 \in x^*$, $u_0 \in u_0^*$ and $u_1 \in u_1^*$. Next, we pick any polynomial $p(z) = \sum_0^N p_i z^i$, with $p_i \in I_i$, and carry out the previous procedure, to obtain an upper bound C for $\|p - f\|$. It is clear then that $f \in \mathcal{U}(I_0, \ldots, I_N; C, 0; 0)$, since, if $f = \sum a_n z^n$, then

$$\sum_{n>N} |a_n| \leq \|f - p\| \leq C$$

QED

Remark: Note that the previous algorithm enables us to construct a neighborhood of type 2 that contains u as a function of z.

Algorithm 3.3: *Given disjoint intervals x_0^* and x_1^*, and representable u_0 and u_1, we construct intervals y_0^* and y_1^* such that the solutions u to (3.3) with initial values u_0 and u_1 for $x \in x_0^*$ satisfy*

$$u(x') \in y_0^* \qquad u'(x') \in y_1^*$$

for any $x' \in x_1^*$.

Description: Choose a representable r such that $r \geq |x_0^* - x_1^*|$, (if we can't, we report a failure) and run the previous algorithm for this r. Then, y_0^* can be readily obtained by simply evaluating the neighborhood $\mathcal{U}$ produced by the algorithm at the interval x_1^*.

In order to obtain y_1^*, we note that

$$u'(x') = u_1 + \int_x^{x'} u^{3/2}(s) s^{-1/2}\, ds$$

and this can be also easily computed. For a sharp bound, note that by the previous remark, we have $u(s)$[as a function of $z = \frac{s-x}{r}$] $\in \mathcal{U}(I_0, \ldots, I_N; 0, C; 2)$, and thus, we also have

$$u(s)^{3/2} s^{-1/2} [\text{as a function of } z] \in \mathcal{U}(I_0, \ldots, I_N; C_h, C_g; 2)$$

After integration, this reduces general error terms by a factor 3 compared to the ones that would follow from the weaker statement

$$u(s)^{3/2} s^{-1/2} [\text{as a function of } z] \in \mathcal{U}(I_0, \ldots, I_N; C_h, C_g; 0)$$

QED

The following lemma has a trivial proof.

Lemma 3.4: *Say y_1 and y_2 are positive solutions of $y'' = x^{-1/2} y^{3/2}$ on the interval $[x_1, x_2]$, with $x_1 > 0$.*

1. *If $y_1(x_1) \geq y_2(x_1)$ and $y_1'(x_1) \geq y_2'(x_1)$ for all $x \in [x_1, x_2]$, then we have that $y_1(x) \geq y_2(x)$ and $y_1'(x) \geq y_2'(x)$ for all $x \in [x_1, x_2]$.*

2. *If $y_1(x_2) \geq y_2(x_2)$ and $y_1'(x_2) \leq y_2'(x_2)$ for all $x \in [x_1, x_2]$, then we have that $y_1(x) \geq y_2(x)$ and $y_1'(x) \leq y_2'(x)$ for all $x \in [x_1, x_2]$.*

Definition: *Let $x_i^* = [x_i^{\mathrm{dn}}, x_i^{\mathrm{up}}]$ for $i = 1, 2$ be two intervals. Then, we define*

$$x_1^* \cup_I x_2^* = [\min_{i=1,2} x_i^{\mathrm{dn}}, \max_{i=1,2} x_i^{\mathrm{up}}]$$

Algorithm 3.5: *Given disjoint intervals x_0^* and x_1^*, and intervals u_0^* and u_1^*, we construct intervals y_0^* and y_1^* such that all solutions u to (3.3) with initial values equal to any $u_0 \in u_0^*$ and any $u_1 \in u_1^*$, for any $x \in x_0^*$ are guaranteed to exist as positive solutions on $[x, x']$, and furthermore satisfy*

$$u(x') \in y_0^* \qquad u'(x') \in y_1^*$$

for all $x' \in x_1^*$.

Description: Assume first that $x_0^* < x_1^*$. Say $u_0^* = [u_0^{\mathrm{dn}}, u_0^{\mathrm{up}}]$, and $u_1^* = [u_1^{\mathrm{dn}}, u_1^{\mathrm{up}}]$. Next, run the previous algorithm: first, for $u_0 = u_0^{\mathrm{dn}}$ and $u_1 = u_1^{\mathrm{dn}}$, to obtain intervals w_0^* and w_1^*, and, second, for $u_0 = u_0^{\mathrm{up}}$ and $u_1 = u_1^{\mathrm{up}}$, two obtain intervals z_0^* and z_1^*. Note that if the first algorithm is successful, this implies that all solutions with initial values u_i^{up} and u_i^{dn} for any $x \in x^*$ are well–defined as strictly positive functions all the way up to x', and, by the previous lemma, all other solutions involved will be bounded above and away from zero: this implies that they can all be well-defined as positive functions all the way up to x'. We can then apply the previous lemma again to conclude that we can put

$$y_0^* = w_0^* \cup_I z_0^* \qquad y_1^* = w_1^* \cup_I z_1^*$$

If $x_0^* > x_1^*$, then we run the previous algorithm, first, for $u_0 = u_0^{\mathrm{dn}}$ and $u_1 = u_1^{\mathrm{up}}$, to obtain intervals w_0^* and w_1^*, and, second, for $u_0 = u_0^{\mathrm{up}}$ and $u_1 = u_1^{\mathrm{dn}}$, two obtain intervals z_0^* and z_1^*. It is then clear as before, that we can put

$$y_0^* = w_0^* \cup_I z_0^* \qquad y_1^* = w_1^* \cup_I z_1^*$$

QED

The Initial Value Problem at 0.

Here we will be concerned with the solution to the Initial Value Problem

$$\left.\begin{aligned} u''(x) &= x^{-1/2} u^{3/2}(x) \\ u(0) &= 1 \\ u'(0) &= -w \end{aligned}\right\} \tag{3.10}$$

for $w > 0$.

In this case, the solution to this problem will be in terms of a function $f \in H^1$ satisfying

$$u(x) = 1 - w \cdot r \cdot z^2 + z^3 f(z) \tag{3.11}$$

where $z = (x/r)^{1/2}$ and r is a small positive representable number.

The solution of (3.10) can be viewed as the fixed point of

$$T(u) = 1 + \int_0^x \left(-w + \int_0^t \frac{u^{3/2}(s)}{s^{1/2}}\, ds \right) dt$$

and again T induces in a trivial way an operator $\tilde{T}$ of which f is its fixed point.

Algorithm 3.6: *We deduce conditions on w, r and α under which $\tilde{T}$ is a contraction in $B(0,\alpha)$, and we compute an upper bound for $\left\|\tilde{T}\right\|_{\text{Lip}}$.*

Description: Let $g = \sum a_n z^n$. Consider the operators

$$\begin{aligned} T_1(f) &= -r\,w\,z^2 + z^3 f(z) \\ T_2(g) &= (1+g(z))^{3/2} \\ T_3(g) &= 4r^{3/2} \sum_{n\geq 0} \frac{a_n z^n}{(n+1)(n+3)} \end{aligned} \tag{3.12}$$

It is clear that

$$T(u) = 1 - wx + z^3 (T_3 \circ T_2 \circ T_1)(f) \tag{3.13}$$

and thus, $\tilde{T} = T_3 \circ T_2 \circ T_1$.
Just as in Algorithm 3.1, T_1 is affine with an isometryt as the linear part, $\|T_3\|_{\text{Lip}} \leq \frac{4}{3} r^{3/2}$ and, for T_2, we have

$$\|T_2\|_{\text{Lip}} \leq C_{2.2}(\tfrac{3}{2}, \gamma_0)$$

where, in this case

$$\sup_{\|f\|\leq\alpha} \|T_1(f)\| \leq rw + \alpha \quad \stackrel{\text{def}}{\equiv} \quad \gamma_0$$

We check that $\gamma_0 < 1$; otherwise, the algorithm fails.
Therefore,

$$\left\|\tilde{T}\right\|_{\text{Lip}} \leq \tfrac{4}{3} r^{3/2} C_{2.2}(\tfrac{3}{2}, \gamma_0)$$

Then we check that the upper bound for $\|\tilde{T}\|_{\text{Lip}}$ thus obtained is strictly less than 1; otherwise, the algorithm fails.

Next, note that $\|T_3(g)\| \leq \frac{4}{3} r^{3/2} \|g\|$, which implies

$$\begin{aligned} \left\|\tilde{T}(0)\right\| &\leq \tfrac{4}{3} r^{3/2} \left\|(1 - wrz^2)^{3/2}\right\| \\ &\leq \tfrac{4}{3} r^{3/2} K_{2.2}(\tfrac{3}{2}, wr) \end{aligned}$$

Then we see that $\tilde{T}$ maps $B(0,\alpha)$ into itself provided

$$\tfrac{4}{3} r^{3/2} K_{2.2}(\tfrac{3}{2}, wr) \leq \alpha \left(1 - \left\|\tilde{T}\right\|_{\text{Lip}}\right)$$ QED

Algorithm 3.7: *Given representable w and r we construct a neighborhood*

$$\mathcal{U}(I_0, \ldots, I_N; 0, C_g, 0)$$

such that the solution of (3.10) is well–defined on $[0, r]$ and satisfies

$$u(x) = 1 - wx + z^3 f(z) \qquad z = \left(\frac{x}{r}\right)^{1/2}$$

for $f \in \mathcal{U}$.

Description: Similar to Algorithm 3.2.

Algorithm 3.8: *Given representable w and r, we construct intervals y_0^* and y_1^* such that the solution u of (3.10) satisfies*

$$u(r) \in y_0^* \qquad u'(r) \in y_1^*$$

Description:
y_0^* can be obtained with a trivial variant of Algorithm 3.3, via Algorithm 3.7.
For y_1^*, note that, if we put

$$u^{3/2}(x) = (T_2 \circ T_1) f(z) = \sum_{n \geq 0} a_n z^n$$

then

$$\begin{aligned} u'(r) &= -w + \int_0^r \frac{u^{3/2}(x)}{x^{1/2}}\, dx \\ &= -w + r^{1/2} \sum_{n=0}^{\infty} \frac{2a_n}{n+1} \end{aligned}$$

Note now that in our representation $z = (x/r)^{1/2}$, we have a neighborhood of type 3 containing $u(x)$ as a function of z. We can thus construct another neighborhood of type 3 such that

$$\sum_{n=0}^{\infty} a_n z^n \in \mathcal{U}(I_0, \ldots, I_N; C_h, C_g; 3)$$

Thus,

$$u'(r) \in -w + r^{1/2} \left(\sum_{n=0}^{N} \frac{2I_n}{n+1} \pm \epsilon \right)$$

whenever

$$|\epsilon| \geq \frac{2C_h}{N+2} + \tfrac{1}{2} C_g \qquad \text{QED}$$

Lemma 3.9: *Let u_1 and u_2 be the solutions of (3.10), with values w_1 and w_2, $w_1 < w_2$. Then, assuming that $u_{1,2}(x)$ are well–defined and strictly positive for $x \in [0, R]$, we have that $u_1(x) > u_2(x)$ and $u_1'(x) > u_2'(x)$ for $x \in [0, R]$.*

Proof: Let f_1 and f_2 be associated with u_1 and u_2 as in (3.11). Since

$$u_1(x) \geq 1 - w_1 x - z^3 \|f_1\|$$

and

$$u_2(x) \leq 1 - w_2 x + z^3 \|f_2\|$$

for all x small enough, we have that $u_1(x) > u_2(x)$ and thus $u_1''(x) > u_2''(x)$. Since the u_i'' are integrable at the origin, we conclude that

$$u_1'(x) = \int_0^x u_1''(t)\, dt - w_1 > \int_0^x u_2''(t)\, dt - w_2 = u_2'(x)$$

for all x small enough. The lemma now follows from Lemma 3.4. QED

Algorithm 3.10: *Given representable r and t, and an interval w^*, we construct intervals y_0^* and y_1^* such that any solution u of (3.10) for any $w \in w^*$ can be continued to $[0, t]$ and satisfies*

$$u(t) \in y_0^* \qquad u'(t) \in y_1^*.$$

Description: Run Algorithm 3.8 twice, once for each endpoint of w^*, to obtain two pairs of intervals w_0^*, w_1^* and z_0^*, z_1^*. Lemma 3.9 then shows that all solutions of (3.10) with $w \in w^*$ are bounded above and away from 0, and can thus be extended as well–defined positive functions over $[0,t]$. Then, Lemma 3.9 again allows us to put

$$y_0^* = w_0^* \cup_I z_0^* \qquad y_1^* = w_1^* \cup_I z_1^*$$

QED

The Initial Value Problem at Infinity.
Here we will be concerned with the solution to the Initial Value Problem

$$\left.\begin{aligned} u''(x) &= x^{-1/2} u^{3/2}(x) \\ u(\infty) &= 0 \\ b_1 &= b \end{aligned}\right\} \tag{3.14}$$

where the last condition is interpreted in the sense of (3.2).
The solution to this problem in this case will be expressed as

$$u(x) = \frac{144}{x^3}\left(1 + bx^{-\alpha} + z^2 f(z)\right) \tag{3.15}$$

where $f \in H^1$, $z = R^\alpha x^{-\alpha}$, for some R large. In this case, the operators involved are not so obvious. Define

$$\begin{aligned} T_1(f) &= bR^{-\alpha} z + z^2 f(z) \\ T_2(g) &= (1+g)^{3/2} \\ T_3(g) &= 12 \sum_{n=2}^{\infty} \frac{a_n z^{n-2}}{(n\alpha+3)(n\alpha+4)} \end{aligned}$$

where, in the last formula, $g(z) = \sum_{n\geq 0} a_n z^n$. Then, put

$$\tilde{T} = T_3 \circ T_2 \circ T_1$$

We now check that if f is a fixed point of $\tilde{T}$ in H^1, then u defined as in (3.15) solves (3.14). Note first that

$$\frac{u^{3/2}(x)}{x^{1/2}} = \frac{12 \cdot 144}{x^5}\left(T_2 \circ T_1\right)(f)$$

where

$$(T_2 \circ T_1)(f) = \sum_{n=0}^{\infty} a_n z^n \qquad a_0 = 1 \quad a_1 = \tfrac{3}{2} b R^{-\alpha}$$

Therefore, since u and its derivatives vanish at ∞,

$$\begin{aligned} u(x) &= \int_x^{\infty} \int_r^{\infty} \frac{u^{3/2}(t)}{t^{1/2}} \, dt \, dr \\ &= \frac{144}{x^3} \left(1 + \frac{12 \, a_1}{(3+\alpha)(4+\alpha)} z + z^2 \tilde{T}(f) \right) \end{aligned}$$

Since $\alpha = \frac{1}{2}(\sqrt{73} - 7)$ satisfies the equation $(\alpha+3)(\alpha+4) = 18$, u satisfies (3.14).

The problem here is considerably more subtle than in the previous cases, due to the fact that T_3 does not scale with R. As a consequence, contraction properties of $\tilde{T}$ either hold or don't, and taking large R won't help much. We are lucky, however, that the norm of T_2 is essentially $\frac{3}{2}$, and that the norm of T_3 is essentially

$$\frac{12}{(2\alpha+3)(2\alpha+4)} < \tfrac{1}{2}$$

which says that the Lipschitz norm of $\tilde{T}$ will approximately be $\frac{3}{4}$. We make this precise now.

Lemma 3.11: *Put $\beta = 0.3$. Assume that $\left|\bar{b}\right| = R^{-\alpha}|b| \leq 0.23$. Then $\tilde{T}$ is a contraction in $B(0,\beta)$, and $\left\|\tilde{T}\right\|_{\mathrm{Lip}} \leq 0.8652$.*

Proof: (Calculator–Assisted) Let f_1, $f_2 \in B(0,\beta)$, and put $\bar{f} = f_1 - f_2$.

$$(T_2 \circ T_1)(f) = 1 + \tfrac{3}{2}\left(\bar{b}z + z^2 f\right) + \tfrac{3}{8}\left(\bar{b}^2 z^2 + 2\bar{b}z^3 f + z^4 f^2\right) + \sum_{n \geq 3} \binom{\frac{3}{2}}{n} \left(\bar{b}z + z^2 f\right)^n$$

So,

$$(T_2 \circ T_1)(f_2) - (T_2 \circ T_1)(f_2) = \tfrac{3}{2} z^2 \bar{f} + \tfrac{3}{4}\bar{b}z^3 \bar{f} + \tfrac{3}{8}\left(z^4 (f_1^2 - f_2^2)\right)$$

$$+ \sum_{n \geq 3} \binom{\frac{3}{2}}{n} \left(\left(\bar{b}z + z^2 f_1\right)^n - \left(\bar{b}z + z^2 f_2\right)^n \right)$$

Now, since T_3 is linear, bounded, and the sum converges absolutely, we have

$$\begin{aligned}\tilde{T}(f_1) - \tilde{T}(f_2) = \tfrac{3}{2}T_3(z^2\bar{f}) + \tfrac{3}{4}\bar{b}T_3(z^3\bar{f}) + \tfrac{3}{8}T_3\left(z^4(f_1^2 - f_2^2)\right) \\ + \sum_{n\geq 3}\binom{\frac{3}{2}}{n}T_3\left(\left(\bar{b}z + z^2 f_1\right)^n - \left(\bar{b}z + z^2 f_2\right)^n\right)\end{aligned}$$

Note now that, for any $f \in H^1$, we have

$$\left\|T_3(z^k f)\right\| \leq \frac{12}{(k\alpha + 3)(k\alpha + 4)}\left\|z^k f\right\|$$

and that

$$\begin{gathered}\left(\bar{b}z + z^2 f_1\right)^n - \left(\bar{b}z + z^2 f_2\right)^n = z^{n+1}h(z) \\ \left\|\left(\bar{b}z + z^2 f_1\right)^n - \left(\bar{b}z + z^2 f_2\right)^n\right\| \leq n\left\|f_1 - f_2\right\|(|\bar{b}| + \beta)^{n-1}\end{gathered}$$

Thus,

$$\begin{aligned}\left\|\tilde{T}\right\|_{\mathrm{Lip}} &\leq \frac{3}{2}\frac{12}{(2\alpha+3)(2\alpha+4)} + \frac{3|\bar{b}|}{4}\frac{12}{(3\alpha+3)(3\alpha+4)} + \frac{3}{8}\frac{12\cdot 2\cdot\beta}{(4\alpha+3)(4\alpha+4)} \\ &\quad + \sum_{n\geq 3}\left|\binom{\frac{3}{2}}{n}\right|\frac{12n}{\left((n+1)\alpha+3\right)\left((n+1)\alpha+4\right)}(|\bar{b}|+\beta)^{(n-1)} \\ &\leq 0.72 + .27|\bar{b}| + 0.21\beta + X + Y\frac{(|\bar{b}|+\beta)^{20}}{1-|\bar{b}|-\beta} \\ &\leq 0.8652\end{aligned}$$

where we have set

$$X = \sum_{n=3}^{20}\left|\binom{\frac{3}{2}}{n}\right|\frac{12n}{\left((n+1)\alpha+3\right)\left((n+1)\alpha+4\right)}(|\bar{b}|+\beta)^{(n-1)}$$

and

$$Y \stackrel{\text{def}}{\equiv} \left|\binom{\frac{3}{2}}{21}\right|\frac{12\cdot 21}{\left(22\alpha+3\right)\left(22\alpha+4\right)} \geq \left|\binom{\frac{3}{2}}{n}\right|\frac{12n}{\left((n+1)\alpha+3\right)\left((n+1)\alpha+4\right)}$$

for $n \geq 21$, and we have used

$$X \leq 0.019 \qquad Y\frac{(|\bar{b}|+\beta)^{20}}{1-|\bar{b}|-\beta} \leq 9\cdot 10^{-10}$$

On the other hand,

$$(T_2 \circ T_1)(0) = \sum_{n \geq 0} \binom{\frac{3}{2}}{n} |\bar{b}|^n z^n$$

Thus,

$$\begin{aligned}
\left\| \tilde{T}(0) \right\| &\leq \sum_{n \geq 2} \left| \binom{\frac{3}{2}}{n} \right| \frac{12 |\bar{b}|^n}{(n\alpha + 3)(n\alpha + 4)} \\
&\leq \tfrac{3}{16} |\bar{b}|^2 + 0.0225 |\bar{b}|^3 + 0.0066 \frac{|\bar{b}|^4}{1 - |\bar{b}|} \\
&\leq 0.01022
\end{aligned}$$

Therefore,

$$\left\| \tilde{T}(f) \right\| \leq 0.01022 + 0.8652\beta \leq \beta$$

and $\tilde{T}$ maps $B(0, \beta)$ into itself. QED

Algorithm 3.12: *Given b^* (interval) and R (representable), we produce $\mathcal{U}_1$ such that, for any $b \in b^*$, the solution u of (3.13) is given by*

$$y(x) = \frac{144}{x^3} \left(1 + b x^{-\alpha} + z^2 f(z)\right) \qquad z = R^\alpha x^{-\alpha}$$

with $f \in \mathcal{U}_1$. Here, $\mathcal{U}_1$ depends only on b^*, i.e., it is independent of which particular b in b^* we are considering.

Description: We first check that we are in the hypothesis of Lemma 3.11. In this case, $\tilde{T}$ has a fixed point f, and, as we saw before, y defined as above satisfies the ODE.

In order to obtain bounds for f, we first look for a heuristic guess p: for example, we iterate $\tilde{T}$ (and truncate) a few times, starting with the function 0. Then, since computing rigorously $\tilde{T}p$ for all $b \in b^*$ poses no difficulty in view of Algorithm 2.3, we conclude that

$$\|f - p\| \leq \frac{\left\| \tilde{T}p - p \right\|}{1 - 0.8652} \leq 7.5 \left\| \tilde{T}p - p \right\| \qquad \text{all } b \in b^*$$

Note that p is the same for all $b \in b^*$, but $\tilde{T}p$ still depends on b. However, the computation of

$$\sup_{b \in b^*} \left\| \tilde{T}p - p \right\|$$

poses no problem, since it is less than or equal to $\left\|\tilde{T}p - p\right\|$ in the interval arithmetic sense.
The algorithm fails if the hypothesis of Lemma 3.11 are not met, or if $\|p\| > 0.3$. QED

Algorithm 3.13: *Given b and R, we produce two intervals u_0^* and u_1^* such that, if u is the solution to* (3.13)*, we have*

$$u(R) \in u_0^* \qquad u'(R) \in u_1^*$$

Description: First, run Algorithm 3.12 for these values of b and R.
Again, it is easy to obtain u_0^*.
Let f be related to u as in (3.15). Then, say

$$\left(1 + bx^{-\alpha} + z^2 f(z)\right)^{3/2} = \sum_{n \geq 0} a_n z^n \in \mathcal{U}(I_0, \ldots, I_N; C_h, C_g; 2)$$

Then,

$$\begin{aligned} u'(R) &= -\int_R^\infty \frac{144 \cdot 12}{x^5} \sum a_n z^n \, dx \\ &= \frac{-144 \cdot 12}{R^4} \sum \frac{a_n}{n\alpha + 4} \\ &\in \frac{-144 \cdot 12}{R^4} \left(\sum_{n=0}^{N} \frac{I_n}{4 + n\alpha} \pm \epsilon \right) \end{aligned}$$

with

$$|\epsilon| \leq \frac{C_h}{4 + (N+1)\alpha} + \frac{C_g}{4 + 2\alpha}$$ QED

Remark: Note that it is enough to run this algorithm for representable values of b, due to the monotonicity of the T–F equation (Lemma 3.4). We omit the trivial details, which are similar to those in Algorithm 3.5

The Boundary Value Problem

Next we discuss how to solve the Boundary Value Problem

$$\left.\begin{aligned} u''(x) &= x^{-1/2} u^{3/2}(x) \\ u(0) &= 1 \\ u(\infty) &= 0 \end{aligned}\right\}$$

We first describe how to obtain bounds for w_0.

Lemma 3.14: *Let u be the solution of (3.3), with $u_1 < 0$. If*

$$\frac{2u_0^{5/2}}{x_0^{1/2}} \leq u_1^2$$

then, there exists a point $t > x_0$ such that u can be extended as a well–defined positive solution of the ODE to $[x_0, t)$ and, furthermore, $\inf_{x \in (x_0, t)} u(x) = 0$.

Proof: Assume the lemma is false. It follows from general ODE considerations that, either u can be extended as a positive well–defined solution of the ODE, or else there exists a T such that $\sup_{x \in (x_0, T)} u(x) = \infty$.
Let

$$d = \frac{|u_1| x_0^{1/2}}{u_0^{3/2}}$$

and note that in both of the two cases above u extends to a well–defined positive solution of the ODE to $(x_0, x_0 + d)$ and furthermore, $u \leq u_0$ on $[x_0, x_0 + d]$. Indeed, consider two cases:

a. u can be extended as a positive solution of the ODE all the way up to ∞. Then, if $u'(x) < 0$ it is trivial. Otherwise, let $x_1 > x_0$ be the first (and only) zero of u'; this means in particular that $u \leq u_0$ on $[x_0, x_1]$. Then, our claim follows by noting that

$$|u_1| \leq \sup_{(x_0, x_1)} u'' \cdot |x_0 - x_1| \leq u_0^{3/2} x_0^{-1/2} |x_0 - x_1|$$

which implies $[x_0, x_0 + d] \subset [x_0, x_1]$.

b. u can be extended as a positive solution of the ODE all the way up to T, where it blows up. Since $u'(x_0) < 0$, there exists x_1, such that $x_0 < x_1 < T$ and $u'(x_1) = 0$. As before, x_1 is the first (and only) zero of u', $u \le u_0$ on $[x_0, x_1]$, and $x_0 + d \le x_1$.

Then, again, since $u'' \le x_0^{-1/2} u_0^{3/2}$ on $[x_0, x_0 + d]$, we conclude that

$$u(x) \le u_0 + u_1(x - x_0) + \frac{u_0^{3/2}}{2x_0^{1/2}}(x - x_0)^2 \qquad x \in [x_0, x_0 + d]$$

The lemma then follows by noting that this parabolic bound attains its minimum at exactly $x_0 + d$, and that this minimum is non–positive if the hypothesis in the statement of the lemma is satisfied. QED

Algorithm 3.15: *Given a representable w, we construct an algorithm that, if successful, will indicate whether $w < w_0$ or $w > w_0$.*

Description: By repeated applications of the previous algorithms, we can determine points x_i and intervals I_i, I'_i, for $i = 0, \ldots, n$, for n large, such that the solution to the TF equation with initial values $u(0) = 1$, $u'(0) = -w$ satisfies $u(x_i) \in I_i$ and $u'(x_i) \in I'_i$. These algorithms also guarantee us that u does not vanish on $[0, x_n]$.

If, for some i, we have $I_i < I_{i+1}$, or $I'_i > 0$, this implies that for some $r_0 < x_n$, u is increasing and convex on $[r_0, r_0 + \epsilon)$, and u will either not vanish at ∞, or blow up and cease to exist at a finite R_0. It is then clear by Lemma 3.9 that $w_0 > w$.

On the other hand, we know that if u becomes arbitrary small on $(0, t)$ for some t, then $w_0 < w$. Using the previous lemma, we then know that, if for some i, we have

$$\frac{2I_i^{5/2}}{x_i^{1/2}} \le |I'_i|^2$$

then we have that $w_0 < w$.
If neither of the above happens, then we quit the algorithm without making any claims for bounds for w_0. QED

Algorithm 3.16: *Assuming bounds for w_0, and given $x_i \in \mathcal{R}$, we can produce y_i^* and ${y'}_i^* \in \mathcal{I}$, $i = 0, \ldots, m$, such that*

$$y(x_i) \in y_i^* \qquad y'(x_i) \in {y'}_i^* \qquad i = 0, \ldots, m$$

Description: Apply Algorithm 3.10 for $r = x_0$, and then iterate Algorithm 3.5 for the x_i. This algorithm will fail if either Algorithm 3.10 or any of the runs of Algorithm 3.5 fails. QED

In order to ensure success for all algorithms, the choice of the x_i will in practice be rather delicate, as will be explained in Section 7.

Lemma 3.17: *Let u_1 and u_2 be the solutions of (3.14) with $b_1 = a_1$ and $b_1 = a_2$ respectively; then, if $a_1 \leq a_2$ and $u_1 > 0$ on $[M, \infty)$, then we have that $u_1(x) \leq u_2(x)$ and $u_1'(x) \geq u_2'(x)$ for all $x \in [M, \infty)$.*

Proof: Obviously it is enough to assume $a_1 < a_2$. Let f_1 and f_2 be the functions associated with the u_i as in (3.15), with R common for the two of them, and large (perhaps a lot larger than M). Then,

$$u_1(x) \leq \frac{144}{x^3}\Big(1 + z(a_1 R^{-\alpha} + z\,\|f_1\|)\Big)$$

and

$$u_2(x) \geq \frac{144}{x^3}\Big(1 + z(a_2 R^{-\alpha} - z\,\|f_2\|)\Big)$$

Now, take R large so

$$a_1 R^{-\alpha} + \|f_1\| < a_2 R^{-\alpha} - \|f_2\|$$

$$a_1 R^{-\alpha} + \|f_1\| < 1$$

This ensures that $0 < u_1(x) < u_2(x)$ for $x > R$ and thus $u_1''(x) < u_2''(x)$ for all $x > R$. Now, note that

$$u_i'(x) = -\int_x^\infty u_i''(t)\,dt \qquad \text{for } i = 1, 2$$

which implies that, not only do we have $0 < u_1(x) < u_2(x)$ for $x > R$, but also $0 > u_1'(x) > u_2'(x)$ for all $x > R$. Finally, if R is larger than M, we apply Lemma 3.4 to guarantee that $0 < u_1(x) \leq u_2(x)$ and $u_1'(x) \geq u_2'(x)$ for $x \in [M, R]$ and thus for all $x \geq M$. QED

Algorithm 3.18: *Assuming bounds for b_1, we can produce $x_i \in \mathcal{R}$, and y^*_i, ${y'}^*_i \in \mathcal{I}$, $i = 1, \ldots, m$, such that*

$$y(x_i) \in y^*_i \qquad y'(x_i) \in {y'}^*_i \qquad i = 1, \ldots, m$$

Description: We choose the x_i in increasing order in i. We apply Algorithm 3.13 and Lemma 3.17 for $R = x_m$, and then iterate —going backwards— Algorithm 3.5 for the x_i. This algorithm will fail if either Algorithm 3.13 or any of the runs of Algorithm 3.5 fails. QED

Remark: Strictly speaking, the choice of the x_i above is purely heuristic, and any choice yields a rigorous answer. In practice, most choices of x_i will yield as an answer "`failure`", which, although completely rigorous (after all, no theorem is claimed), is not very useful. As a result, it is important to make a good choice of the x_i. In practice, these x_i will be the same as the one used in Algorithm 3.16, whose choice is explained in Section 7.

Algorithm 3.19: *Given a representable b, and assuming bounds for w_0, we construct an algorithm that, if successful, will indicate whether $b < b_1$ or $b > b_1$.*
Also, assuming bounds for b_1, and given w, we indicate whether $w < w_0$ or $w > w_0$.

Description: Let y be the Thomas–Fermi function, and u be the solution of (3.13). Assuming bounds for w_0, Algorithm 3.16 allows us to produce representable x_i and intervals I_i and I'_i, such that $y(x_i) \in I_i$ and $y'(x_i) \in I'_i$. For these x_i, using Algorithm 3.13 and repeated applications of Algorithm 3.5 (going backwards), we can produce intervals J_i and J'_i such that $u(x_i) \in J_i$ and $u'(x_i) \in J'_i$. In this situation we can again guarantee that $u > 0$.
Then, if for some i

$$I_i > J_i \qquad \text{or} \qquad I'_i < J'_i$$

then we have $b < b_1$. If, however, we have

$$I_i < J_i \qquad \text{or} \qquad I'_i > J'_i$$

then we have $b > b_1$.
We report a failure if

$$I_i \cap J_i \neq \emptyset \qquad I'_i \cap J'_i \neq \emptyset$$

for all i, in which case no relation is claimed between b and b_1.

The rest of the algorithm follows along the same lines. QED

Note that the last part of the previous algorithm constitutes a refinement of Algorithm 3.16, but it requires bounds for b_1. Also, Algorithm 3.15 allows us to obtain an initial, probably wasteful, bound for w_0. This initial bound allows us to obtain a bound for b_1, which in turn will allow us to improve our initial bound for w_0. Iterating this last algorithm in this way allows us to obtain improved bounds for both w_0 and b_1. The intersection of the bounds produced by Algorithms 3.16 and 3.18 are improved bounds for the Thomas–Fermi function and its derivative at points x_i. These translate immediately to better bounds for the solution of the Thomas–Fermi equation, and related constants.

Algorithm 3.20: *We can produce x_i, $r_i \in \mathcal{R}$, and*

$$\mathcal{U}_i(I_0^i, \ldots, I_N^i; C_{h,i}, C_{g,i}, 2) \qquad i = 1, \ldots, m$$

such that

$$y(x_i + z \cdot r_i) \in \mathcal{U}_i(I_0^i, \ldots, I_N^i; C_{h,i}, C_{g,i}, 2) \qquad i = 1, \ldots, m$$

and

$$\cup_{i=1}^m (x_i - r_i, x_i + r_i) = (x_1 - r_1, x_m + r_m) \subset (0, \infty)$$

Description: Our previous remark gives us the x_i, r_i, I_0^i and I_1^i. The rest follows by applying Algorithm 3.2 for every i.

Lemma 3.21: *The following inequalities hold:*

$$1.588071022611278 \leq w_0 \leq 1.588071022611471$$

$$-13.270973847925352 \geq b_1 \geq -13.270973848125353$$

$$0.486348538043594 \leq \Omega_c^2 \leq 0.486348538046869$$

$$2.104025280219502 \leq r_c \leq 2.104025280273837$$

Needless to say, the decimal numbers quoted above stand for the exact rational numbers they represent.

Proof (Computer–Assisted): The inequalities for w_0 and b_1 follow by carrying out previous algorithms.
The inequality for r_c follows by checking that

$$u'(2.104025280219502) \geq 0 \geq u'(2.104025280273837)$$

The bounds for Ω_c are then trivial. QED

4. Zone I.

The purpose of this section is to prove (1.2) for all Ω in Zone I, as defined at the end of Section 1. We will do this as follows:

First, we partition Zone I into "fat" intervals $\{W_i\}_{i=1}^{n}$. Note that the first such interval will have the form $(0, \Omega_\epsilon]$, for an Ω_ϵ to be picked (much) later in our proof. In fact, the role of the W_i will change as they approach zero: the larger ones (most of them, by the way) will receive identical treatment. Then, there will be a family of them, rather close to zero, which will receive a sort of special treatment, and then the single $W_1 = (0, \Omega_\epsilon]$ which will be on its own.
Second, each fat interval W is divided into a finite partition of (lots of) suitably small subintervals Ω^* (except W_1 which will be both a "fat" and "thin" interval at the same time.) Our aim is to produce uniform bounds for $-F''(\Omega)$ for all $\Omega \in \Omega^*$: for W_1 we will be able to produce only lower bounds, since $-F''$ is unbounded there; for the others, we will be able to produce both upper and lower bounds.

Say $\Omega^* = [z_1, z_2]$ is contained in the fat interval $W = [w_1, w_2]$.
We construct two functions $a(\Omega^*)$ and $b(\Omega^*)$, constant on each subinterval Ω^*, such that

$$r_1(\Omega) < a < b < r_2(\Omega) \qquad \Omega \in \Omega^*$$

In practice, a and b will be very close to r_1 and r_2 respectively.
Now, we recall Lemma 1.2; our job is then to compute each of the following

$$I_1 = \int_a^b \left(u(r) - \Omega^2\right)^{-3/2} y(r)\, dr \tag{4.1a}$$

$$I_2 = \lim_{\delta\to 0} \left(\int_{r_1(\Omega)+\delta}^{a} \left(u(r) - \Omega^2\right)^{-3/2} y(r)\, dr - G_1(\Omega)\delta^{-1/2} \right) \tag{4.1b}$$

$$I_3 = \lim_{\delta\to 0} \left(\int_{b}^{r_2(\Omega)-\delta} \left(u(r) - \Omega^2\right)^{-3/2} y(r)\, dr - G_2(\Omega)\delta^{-1/2} \right) \tag{4.1c}$$

with G_i such that the limit is finite.

The computation of I_1 is done as follows:
Break up

$$I_1 = \sum_{i=1}^{n} \int_{t_i}^{t_{i+1}} \left(u(r) - \Omega^2\right)^{-3/2} y(r)\, dr = \sum_{i=1}^{n} J_i(\Omega)$$

where $t_1 = a$ and $t_{n+1} = b$.
Note that each J_i can be computed directly, since it involves only elementary operations. However, computing *all* J_i like that will take a very long time. To remedy this, we do as follows:
First, we take two numbers $\tilde{a}(\Omega^*)$ and $\tilde{b}(\Omega^*)$, constant on each subinterval Ω^*, such that

$$\tilde{a} = t_{i_0} \qquad \tilde{b} = t_{i_1}$$

with $1 \le i_0$ and $i_1 \le n$. Normally, we will have that $i_0 \le i_1$. It could happen, however, that $i_0 > i_1$ meaning that the computation of the J_i is always done directly, without using the faster method below.
Then, we take t_i for $i = i_0, \ldots, i_1$ to be the same for all $\Omega^* = [z_1, z_2] \subset W = [w_1, w_2]$, and we compute once and for all the following numbers:

$$a_{k,i} = \int_{t_i}^{t_{i+1}} \left(u(r) - w_k^2\right)^{-3/2} y(r)\, dr \qquad b_{k,i} = 3w_k \int_{t_i}^{t_{i+1}} \left(u(r) - w_k^2\right)^{-5/2} y(r)\, dr$$

for $k = 1, 2$, and $i = i_0, \ldots, i_1$.
Next, note that the functions

$$f_i(w) = \int_{t_i}^{t_{i+1}} \left(u(r) - w^2\right)^{-3/2} y(r)\, dr$$

are increasing and convex on W. Therefore, if $w \in [w_1, w_2]$,

$$\max_{k=1,2} \left(f_i'(w_k)\,(w - w_k) + f_i(w_k)\right) \le f_i(w) \le \frac{f_i(w_1) - f_i(w_2)}{w_1 - w_2}\,(w - w_1) + f_i(w_1)$$

Thus,

$$\max_{k=1,2} \left(b_{k,i}\,(\Omega - w_k) + a_{k,i}\right) \le J_i(\Omega) \le \frac{a_1 - a_2}{w_1 - w_2}\,(\Omega - w_1) + a_1 \qquad \Omega \in W$$

This gives us intervals $\tilde{J}_i(\Omega^*)$, such that

$$J_i(\Omega) \subset \tilde{J}_i(\Omega^*) \qquad i_0 \le i \le i_1 \qquad \Omega \in \Omega^* \tag{4.2}$$

In practice, $\tilde{a}$ and $\tilde{b}$ will be far from r_1 and r_2. They will enclose a region which is safely away from the singularities of the integrand in our formula for F'', for which we can expect (4.2) to be sharp.

For i outside of the range $[i_0, i_1]$, we compute $f_i(z_1)$ and $f_i(z_2)$ directly, and, by our previous remark,

$$J_i(\Omega) \in \tilde{J}_i(\Omega^*) \quad \stackrel{\text{def}}{\equiv} \quad [f_i(z_1), f_i(z_2)]$$

Thus, we have defined $\tilde{J}(\Omega^*)$ for all $i = 1, \ldots, n$, and we conclude that

$$I_1(\Omega) \in \sum_{i=1}^{n} \tilde{J}_i(\Omega^*) \qquad \Omega \in \Omega^*$$

Computation of $\mathbf{I_2}$. Consider a small number $\bar{\Omega}_2 \ll \Omega_c$, that we can make coincide with one of the endpoints of the fat intervals W_i.
We distinguish two cases: $\Omega > \bar{\Omega}_2$ and $\Omega \le \bar{\Omega}_2$.

If $\Omega > \bar{\Omega}_2$, we use Algorithm 3.2 to compute $\mathcal{U}_1$ such that

$$u(x) = \Omega^2 + z f(z) \qquad z = \frac{x - r_1(\Omega)}{r} \qquad f \in \mathcal{U}_1$$

where $r \ge |a - r_1(\Omega)|$ and $\mathcal{U}_1$ is uniform for all $\Omega \in \Omega^*$. Note that $f(0) > 0$. Also, in order to apply Algorithm 3.2, we need to obtain bounds for $r_1(\Omega)$ and for $u'(r_1)$; the first can be done by obtaining heuristic bounds r_{dn} and r_{up}, and checking that

$u(r_{\mathrm{dn}}) \leq \Omega^2$ and $u(r_{\mathrm{up}}) \geq \Omega^2$, which can be easily checked using the information given by Algorithm 3.20. Bounds for $u'(r_1)$ can be obtained using the bounds for r_1 and the information on y_{TF} (hence on u) given by Algorithm 3.20. See the section on implementation for more details.
Therefore,

$$\begin{aligned}\int_{r_1(\Omega)+\delta}^{a} \left(u(x)-\Omega^2\right)^{-3/2} y(x)\,dx &= \int_{r_1(\Omega)+\delta}^{a} z^{-3/2} f^{-3/2}(z) y(x)\,dx \\ &= \int_{r_1(\Omega)+\delta}^{a} z^{-3/2} \tilde{f}(z)\,dx\end{aligned}$$

for a new function $\tilde{f}(z) = y(x) f^{-3/2}(z)$, that can also be enclosed in a computable $\mathcal{U}_2$. Note that $\tilde{f}(0) > 0$ also. Thus, if

$$\tilde{f}(z) = \sum_{n\geq 0} a_n z^n \in \mathcal{U}(J_0, \ldots, J_N; C_h, C_g; 1)$$

we see that

$$\begin{aligned}\int_{r_1(\Omega)+\delta}^{a} \left(u(x)-\Omega^2\right)^{-3/2} y(x)\,dx &= \int_{r_1(\Omega)+\delta}^{a} \sum_{n\geq 0} a_n z^{n-\frac{3}{2}}\,dx \\ &= r \sum_{n\geq 0} \frac{a_n}{n-\frac{1}{2}} z^{n-\frac{1}{2}} \Bigg|_{z=\frac{\delta}{r}}^{z=\frac{a-r_1(\Omega)}{r}}\end{aligned}$$

This implies that

$$\begin{aligned}I_2 &= r \sum_{n\geq 0} \frac{a_n}{n-\frac{1}{2}} \left(\frac{a-r_1(\Omega)}{r}\right)^{n-\frac{1}{2}} \\ &\in r \sum_{n=0}^{N} \frac{J_n}{n-\frac{1}{2}} \left(\frac{a-r_1(\Omega)}{r}\right)^{n-\frac{1}{2}} \pm \epsilon\end{aligned}$$

with

$$|\epsilon| \leq r \left(\frac{C_h}{N+\frac{1}{2}} + 2C_g\right)$$

When $\Omega \leq \bar{\Omega}_2$, we proceed as follows:

Consider the change of variables given by $r(t)$, the inverse of u. Then, by the last remark in Lemma 1.2,

$$I_2 = \frac{d}{d\Omega}\left(\Omega \int_{r_1(\Omega)}^{a} \left(u(r) - \Omega^2\right)^{-1/2} \frac{dr}{r}\right) = \frac{d}{d\Omega}\left(\Omega \int_{\Omega^2}^{u(a)} \left(t - \Omega^2\right)^{-1/2} w(t)\, dt\right)$$

for

$$w(t) = \frac{r'(t)}{r(t)}$$

In order to compute w, we consider the following:

Let x_0 and ϵ' be small numbers satisfying

a. $u'(x) > 1 - \epsilon'$ for $x \in [0, x_0]$.

b. For a sequence $\{\bar{b}_n\} \in l^1$, we have

$$u(x) = x\left(1 + \sum_{n=2}^{\infty} \bar{b}_n \bar{x}^{n/2}\right) \qquad x \leq x_0$$

Furthermore, we know that

$$1 + \sum_{n=2}^{\infty} \bar{b}_n z^n \in \mathcal{U}(I_0, \ldots, I_m; 0, C_g; 2) \tag{4.3}$$

with $I_0 = [1,1]$ and $I_1 = [0,0]$. Here, $\bar{x}$ denotes x/x_0, and $\bar{b}_n = b_n \cdot x_0^{n/2}$.

Define

$$\bar{t} = \left(\frac{t}{x_0}\right)^{1/2}$$

We also consider a small number $\eta \leq u(x_0)$. It will be chosen so that (4.10) below holds. We start by obtaining expressions for $u'(r)$ and $u''(r)$ similar to the one for u in (4.3). Note first that (4.3) is equivalent to an expression for $y(x)$. Then, by the Thomas–Fermi equation, and by integration, we have

$$y''(x) = x^{-1/2}\left(\sum_{n=0}^{\infty} \bar{b}_n \bar{x}^{n/2}\right)^{3/2} = x^{-1/2} \sum_{n=0}^{\infty} y''_n \bar{x}^{n/2} = x^{-1/2} y_{pp}(\bar{x}^{1/2}) \qquad y''_0 = 1$$

$$y'(x) = \sum_{n=0}^{\infty} y'_n \bar{x}^{n/2} = y_p(\bar{x}^{1/2}) \qquad y'_n = \begin{cases} -w_0 & n = 0 \\ \frac{2}{n} x_0^{1/2} y''_{n-1} & n > 0 \end{cases}$$

from which we obtain

$$u'(x) = xy'(x) + y(x) = \sum_{n=0}^{\infty} u'_n \bar{x}^{\,n/2} = u_p(\bar{x}^{1/2}) \tag{4.4a}$$

$$u'_n = \begin{cases} 1 & n = 0 \\ 0 & n = 1 \\ \bar{b}_n + y'_{n-2} \cdot x_0 & n \geq 2 \end{cases}$$

$$u''(x) = xy''(x) + 2y'(x) = u_{pp}(\bar{x}^{1/2}) = \sum_{n=0}^{\infty} u''_n \bar{x}^{\,n/2} \tag{4.4b}$$

$$u''_n = \begin{cases} -2w_0 & n = 0 \\ 2y'_n + x_0^{1/2} y''_{n-1} & n > 0 \end{cases}$$

Note that since we know a neighborhood of type 2 that contain y, we can enclose y_{pp} and u_p also in neighborhoods of type 2, and y_p and u_{pp} in neighborhoods of type 3.

We start our analysis understanding $r(t)$.
First, a technical algorithm.

Algorithm 4.1: *Given a function $r(t) = t \cdot R(\bar{t})$ with*

$$R(z) \in \mathcal{U}^0(a_0^*, \ldots, a_N^*; C_N, 0; \infty; S) \qquad a_0^* = [1,1]$$

valid for $\bar{t} \in S \subset [0,1]$, and given a neighborhood $\mathcal{U}^1$ in H^1, we can compute another neighborhood $\mathcal{U}_2^0$, also of type ∞ in C^0, also valid on $\bar{t} \in S$, such that if

$$f(t) = \sum_{n=0}^{\infty} \bar{c}_n \bar{t}^{\,n} \in \mathcal{U}^1(I_0, \ldots, I_N; C_h, C_g, m)$$

then $G(\bar{t}) \in \mathcal{U}_2^0$ for

$$G(\bar{t}) = f\big(r(t)\big) = \sum_{n=0}^{\infty} \bar{c}_n \left(\frac{r(t)}{x_0}\right)^{n/2}$$

We assume that $0 < r(t) \leq x_0$ for $\bar{t} \in S$.

Description: Consider any $a_i \in a_i^*$. Put

$$\left(\sum_{n=0}^{N} a_n \bar{t}^{\,n} + \bar{t}^{N+1} h(\bar{t})\right)^{\gamma} = \sum_{n=0}^{n_0} a_{n,\gamma} \bar{t}^{\,n} + \bar{t}^{\,n_0+1} h(\bar{t}; \gamma, n_0 + 1)$$

with $a_{n,\gamma} \in a^*_{n,\gamma}$, the $a^*_{n,\gamma}$ easily determined intervals,

$$\|h(\bar{t};\gamma,n_0)\|_{C^0} \le \epsilon_{\gamma,n_0}.$$

We can see, on one hand, that

$$\begin{aligned}\sum_{n=0}^{N} \bar{c}_n \left(\frac{r(t)}{x_0}\right)^{n/2} &= \sum_{n=0}^{N} \bar{c}_n \bar{t}^{\,n} \left(\sum_{k=0}^{N-n} a_{k,n/2} \bar{t}^{\,k} + \bar{t}^{\,N+1-n} h(\bar{t};\tfrac{k}{2},N-n)\right) \\ &= \sum_{n=0}^{N} d_n \bar{t}^{\,n} + \bar{t}^{\,N+1} h(t)\end{aligned}$$

with

$$\begin{aligned}|h(t)| &\le \sum_{n=0}^{N} |\bar{c}_n|\, \epsilon_{\frac{n}{2},N-n+1} \\ &\le \sum_{n=0}^{N} |I_n|\, \epsilon_{\frac{n}{2},N-n+1} + C_g \max_{n=m,\ldots,N} \epsilon_{\frac{n}{2},N-n+1}\end{aligned}$$

and

$$\begin{aligned}d_n &= \sum_{i+j=n} \bar{c}_i \cdot a_{j,\frac{i}{2}} \\ &\in \sum_{i+j=n} I_i \cdot a^*_{j,\frac{i}{2}} \pm \epsilon_n \quad \stackrel{\text{def}}{\equiv} \quad d^*_n\end{aligned}$$

where

$$\epsilon_n \le \begin{cases} C_g \sup_{m\le i\le n} \left|a_{n-i,\frac{i}{2}}\right| & \text{if } n \ge m \\ \\ 0 & \text{otherwise}\end{cases}$$

On the other hand, since

$$\begin{aligned}\sum_{n=N+1}^{\infty} |\bar{c}_n| \left(\frac{r(t)}{x_0}\right)^{n/2} &\le (C_g + C_h)\left(\frac{r(t)}{x_0}\right)^{\frac{(N+1)}{2}} \\ &\le (C_g + C_h)\, \bar{t}^{\,N+1} \left(1 + \sum_{n=1}^{N} |a_n| + C_N\right)^{\frac{(N+1)}{2}}\end{aligned}$$

we conclude that

$$f(r(t)) = G(\bar{t}) \in \mathcal{U}^0(d_0^*, \ldots, d_N^*; \tilde{C}_h, 0; \infty)$$

with

$$\tilde{C}_h = (C_g + C_h)\left(1 + \sum_{n=1}^{N} |a_n| + C_N\right)^{\frac{(N+1)}{2}} + \sum_{n=0}^{N} |I_n|\, \epsilon_{n/2, N-n+1}$$
$$+ C_g \max_{n=m,\ldots,N} \epsilon_{n/2, N-n+1} \qquad \text{QED}$$

Algorithm 4.2: *Given $N \geq 0$, we produce intervals $a_2^*, \ldots, a_N^*$, and a constant C_N, such that*

$$\left| r(t) - t\Big(1 + \sum_{n=2}^{N} a_n \bar{t}^{\,n}\Big) \right| \leq C_N\, t \cdot \bar{t}^{\,N+1}$$

for constants $a_i \in a_i^*$, $i = 2, \ldots, N$, and for $t \leq \eta$.

Description: First, we will construct an inductive procedure to define numbers a_n such that

$$u\left(r_N(t)\right) = t\left(1 + O\big(\bar{t}^{\,N+1}\big)\right) \qquad t \to 0 \tag{4.5}$$

where

$$r_N(t) = t\left(1 + \sum_{n=2}^{N} a_n \bar{t}^{\,n}\right)$$

By induction. For $N = 1$, let $r_0(t) = t$. Note that $r(t) \geq r_0(t)$, that $r_0(t) \leq x_0$ for $\bar{t} \leq 1$, and that, if $t \leq \eta \leq u(x_0)$ then $\bar{t} \leq 1$.
Therefore, we have

$$u(r_0(t)) = t\left(1 + \sum_{n \geq 2} b_n t^{n/2}\right)$$

Thus,

$$|r_0(t) - r(t)| \leq t\, \frac{\left|\sum_{n\geq 2} b_n t^{n/2}\right|}{\inf_{r \in [r_0(t)\,,\, r(t)]} |u'(r)|} \tag{4.6}$$

Since, for t small enough, $r(t) \leq x_0$, the denominator is bounded below by $(1 - \epsilon')$, and we conclude

$$r_0(t) - r(t) = O\big(t^2\big)$$

For general N, we set

$$r_N(t) = t\left(1 + \sum_{n=2}^{N} a_n \bar{t}^{\,n}\right)$$

where $a_2, \ldots, a_{N-1}$ satisfy the induction hypothesis.
Note that we have

$$\sum_{n=1}^{\infty} b_n r_N(t)^{n/2} - \sum_{n=1}^{\infty} b_n r_{N-1}(t)^{n/2} = O\left(\bar{t}^{\,N+1}\right) \tag{4.7}$$

Thus, if for any real number γ we put

$$\left(1 + \sum_{n=2}^{N-1} a_n \bar{t}^{\,n}\right)^{\gamma} = \sum_{n=0}^{N} a_{n,\gamma} \bar{t}^{\,n} + O\left(\bar{t}^{\,N+1}\right) \tag{4.8}$$

we see that

$$\begin{aligned} u(r_{N-1}(t)) &= t\left(1 + \sum_{n=2}^{N-1} a_n \bar{t}^{\,n}\right) \\ &\quad \cdot \left(\sum_{n=0}^{N} \bar{b}_n \bar{t}^{\,n} \left(\sum_{i=0}^{N} a_{i,n/2} \bar{t}^{\,i} + O\left(\bar{t}^{\,N+1}\right)\right) + O\left(\bar{t}^{\,N+1}\right)\right) \\ &= t\left(1 + \sum_{n=2}^{N-1} a_n \bar{t}^{\,n}\right) \cdot \left(\sum_{n=0}^{N} c_n \bar{t}^{\,n} + O\left(\bar{t}^{\,N+1}\right)\right) \end{aligned} \tag{4.9}$$

where

$$c_k = \sum_{n=0}^{k} \bar{b}_n \cdot a_{k-n,n/2} \qquad c_0 = 1.$$

By the induction hypothesis, (4.9) is equal to $t(1 + O(\bar{t}^{\,N}))$. Thus, using (4.7), we can see that

$$u(r_N(t)) = t\left(1 + \sum_{n=2}^{N-1} a_n \bar{t}^{\,n} + a_N \bar{t}^{\,N}\right) \cdot \left(\sum_{n=0}^{N} c_n \bar{t}^{\,n} + O\left(\bar{t}^{\,N+1}\right)\right)$$

where the c_n here are the same as those in (4.9).
Therefore, by putting

$$a_N = -\sum_{k=1}^{N-2} a_{N-k} c_k - c_N$$

we get rid of all $\bar{t}^{\,N}$ terms, thus obtaining (4.5).

So far, we have proved the existence of numbers a_n such that (4.5) is satisfied.
This procedure also gives us an algorithm to compute the a_n^*. Indeed, bounds $a_{n,\gamma}^*$ for the $a_{n,\gamma}$ can be computed explicitly, since by the induction hypothesis we already know a_i^*, for $i = 1, \ldots, N-1$. As for the c_k, recalling (4.3),

$$c_k = \sum_{n=0}^{k} \bar{b}_n \cdot a_{k-n,{}^{n}\!/\!{}_{2}} \in \sum_{n=0}^{k} I_n \cdot a_{k-n,{}^{n}\!/\!{}_{2}}^* \pm \epsilon_k$$

with

$$|\epsilon_k| \leq \begin{cases} \sup_{k \geq n \geq 2} |a_{k-n,{}^{n}\!/\!{}_{2}}| \cdot C_g & \text{if } k \geq 2 \\ 0 & \text{if } k \leq 1 \end{cases}$$

In particular, it follows immediately that $a_2 = -\bar{b}_2 = x_0 \cdot w_0 \in -I_2$.

To obtain a good value for the constant C_N, we proceed as follows:
First, check that

$$\eta \cdot \left(1 + \sum_{n=2}^{N} |a_n|\right) \leq x_0 \tag{4.9a}$$

(See also (4.10) below.) This allows us to invoke Algorithm 4.1, with $S = [0, \sqrt{\eta/x_0}]$, to obtain

$$y(r_N(t)) = f(\bar{t}) \in \mathcal{U}_1^0(\cdots;\cdots;\infty)$$

and thus we can write

$$u\,(r_N(t)) = r_N(t) \cdot f(\bar{t}) \equiv t \cdot g(\bar{t})$$

with g belonging to $\mathcal{U}^0(I_0, \ldots, I_N; \tilde{\tilde{C}}_h, 0; \infty; t \leq \eta)$, the product neighborhood of

$$\mathcal{U}_2(a_0^*, \ldots, a_N^*; 0, 0; \infty; t \leq \eta)$$

and $\mathcal{U}_1^0$. Now, since (4.5) implies that $g(\bar{t}) = 1 + O\big(\bar{t}^{\,N+1}\big)$, we can take $I_0 = [1,1]$ and $I_i = [0,0]$, for $i = 1, \ldots, N$. Note that this is relied crucially on the fact that $\mathcal{U}^0$ is of type ∞; in fact, since

$$g(\bar{t}) \in \mathcal{U}^0(I_0, \ldots, I_N; \tilde{\tilde{C}}_h, 0; \infty; t \leq \eta)$$

we can find constants $p_i \in I_i$ such that

$$\left| g(\bar{t}) - \sum_{k=0}^{N} p_i \bar{t}^{\,k} \right| \leq \tilde{\tilde{C}}_h \bar{t}^{\,N+1}$$

On the other hand, since $g(\bar{t}) = 1 + O\big(\bar{t}^{\,N+1}\big)$, we must have $p_0 = 1$ and $p_i = 0$ for $i = 1, \ldots, N$, thus

$$|g(\bar{t}) - 1| \le \tilde{\tilde{C}}_h \bar{t}^{\,N+1}.$$

If Algorithm 4.1 had produced a neighborhood of any other type, it would have been harder to conclude this without changing $\tilde{\tilde{C}}_h$. More precisely, if we have $\phi(t)$ defined on $t \in [-1, 1]$ such that

$$\left|\phi(t) - \sum_{n=0}^{N} a_n t^n\right| \le C t^k \quad \text{and} \quad \phi(t) - \sum_{n=0}^{N} a_n t^n = O\big(t^{N+1}\big)$$

for $t \in [-1, 1]$, we cannot conclude

$$\left|\phi(t) - \sum_{n=0}^{N} a_n t^n\right| \le C t^{N+1}$$

with the same constant C unless $k > N$. Counterexamples with $k \le N$ are readily available (simply take $N = k = 0$, $\phi(t) = t^3 - t$, $a_0 = 0$; then, $|\phi| \le 2^{-1/2}$ but $\phi(t)$ is not bounded by $2^{-1/2}|t|$.)

So, we have that

$$\big|u\big(r_N(t)\big) - t\big| \le \tilde{\tilde{C}}_h \cdot t \cdot \bar{t}^{\,N+1} \qquad t \le \eta$$

Next, note that

$$|r_N(t) - r(t)| \le \frac{|u(r_N(t)) - t|}{\inf_{0 \le r \le \max(r(t), r_N(t))} |u'(r)|}$$

Then, the fact that $\eta \le u(x_0)$ and (4.9a) imply that

$$|u'(r)| \ge (1 - \epsilon') \qquad 0 \le r \le \max\left(r(t), r_N(t)\right) \quad t \le \eta$$

from which the lemma follows by taking

$$C_N = \frac{\tilde{\tilde{C}}_h}{(1 - \epsilon')} \qquad \text{QED}$$

Algorithm 4.3: *We produce a neighborhood* $\mathcal{U}^0(I_0, \ldots, I_N; C_h, 0; \infty)$ *such that*

$$h(t) \overset{\text{def}}{\equiv} t w'(t) + w(t) = f(\bar{t}) \qquad f \in \mathcal{U}$$

for $t \le \eta$.

Description:
We note that
$$r'(t) = \frac{1}{u'\big(r(t)\big)} \qquad r''(t) = -\left(r'(t)\right)^3 u''\big(r(t)\big)$$
Therefore, we check that
$$\eta \cdot \left(1 + \sum_{n=1}^{N} |a_n| + C_N\right) \leq x_0 \tag{4.10}$$
and as a result of this, we can apply Algorithm 4.1 and obtain neighborhoods of type ∞ in C^0 containing functions f, f_p and f_{pp} s.t.
$$r(t) = tf(\bar{t}) \qquad r'(t) = f_p(\bar{t}) \qquad r''(t) = f_{pp}(\bar{t})$$
which are valid for $t \leq \eta$. These functions are obtained by putting
$$f_p(\bar{t}) = \frac{1}{u_p\left(\left(r(t)/x_0\right)^{1/2}\right)} \qquad f_{pp}(\bar{t}) = -f_p^3 \cdot u_{pp}\left(\left(r(t)/x_0\right)^{1/2}\right)$$
Note that with this definition, f and f_p are normalized to be 1 at 0, and $f_{pp}(0) = 2w_0$. Thus,
$$\begin{aligned} w(t) &= \frac{r'(t)}{r(t)} = \frac{1}{t}\left(\frac{f_p(\bar{t})}{f(\bar{t})}\right) \\ w'(t) &= \frac{r''(t)}{r(t)} - \left(\frac{r'(t)}{r(t)}\right)^2 = \frac{1}{t}\left(\frac{f_{pp}(\bar{t})}{f(\bar{t})}\right) - \frac{1}{t^2}\left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2 \end{aligned}$$
and
$$\begin{aligned} h(t) = tw'(t) + w(t) &= t^{-1}\left(\frac{f_p(\bar{t})}{f(\bar{t})} - \left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2\right) + \left(\frac{f_{pp}(\bar{t})}{f(\bar{t})}\right) \\ &= f_h(\bar{t}) \end{aligned}$$
for a function f_h belonging to an easily computable neighborhood of type ∞ in C^0. Note that by our normalization, the t^{-1} terms drop out. Furthermore, there are no $t^{-1}\bar{t}$ terms since neither f nor f_p have $\bar{t}$ terms, and in fact $f_h(0) = w_0$, which we can easily see as follows: first, $f(\bar{t}) = 1 + w_0 t + O\big(\bar{t}^{\,3}\big)$, $f_p(\bar{t}) = 1 + 2w_0 t^2 + O\big(\bar{t}^{\,3}\big)$ and $f_{pp}(\bar{t}) = 2w_0 + O(\bar{t})$, therefore
$$t^{-1}\left(\frac{f_p(\bar{t})}{f(\bar{t})} - \left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2\right) = -w_0 + O\big(t^{-1}\bar{t}^{\,3}\big) = -w_0 + O(\bar{t})$$

and

$$\frac{f_{pp}(\bar{t})}{f(\bar{t})} = 2w_0 + O(\bar{t})$$

thus $f_h(0) = w_0$.
Moreover, if we set

$$\frac{f_p(\bar{t})}{f(\bar{t})} - \left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2 + t \cdot \left(\frac{f_{pp}(\bar{t})}{f(\bar{t})}\right) \in \mathcal{U}^0(I_0, \ldots, I_{N+2}; \epsilon_h, 0; \infty)$$

then

$$f_h \in \mathcal{U}^0(x_0^{-1} \cdot I_2, \ldots, x_0^{-1} I_{N+2}; x_0^{-1} \epsilon_h, 0; \infty; t \leq \eta)$$

valid for $t \leq \eta$.

Now, let

$$f_h(\bar{t}) = \sum_{n=0}^{N} a_n \bar{t}^{\,n} + H(\bar{t})$$

with

$$|H(\bar{t})| \leq \epsilon_h \, |\bar{t}|^{N+1} \qquad t \leq \eta$$

Finally, then, let δ be a small number such that $u(\delta) \leq \eta$, set $\bar{\Omega}_2 \leq \sqrt{u(\delta)}$, and consider $\Omega \leq \bar{\Omega}_2$ for which we set $a(\Omega) \equiv \delta$:

$$\begin{aligned}
\frac{d}{d\Omega}\left(\Omega \int_{r_1(\Omega)}^{\delta} \left(u(r) - \Omega^2\right)^{-1/2} \frac{dr}{r}\right) &= \frac{d}{d\Omega}\left(\Omega^2 \int_1^{\Omega^{-2}u(\delta)} (t-1)^{-1/2}\, w(t\Omega^2)\, dt\right) \\
&= 2\Omega \int_1^{\Omega^{-2}u(\delta)} (t-1)^{-1/2}\, h(t\Omega^2)\, dt \\
&\quad - 2\left(u(\delta) - \Omega^2\right)^{-1/2} w(u(\delta))u(\delta) \\
&= 2\Omega \sum_{n=0}^{N} a_n x_0^{-n/2} \Omega^n \int_1^{\Omega^{-2}u(\delta)} (t-1)^{-1/2}\, t^{\frac{n}{2}}\, dt \\
&\quad + \tilde{h}(\Omega) - \left(u(\delta) - \Omega^2\right)^{-1/2} \frac{2u(\delta)}{\delta u'(\delta)}
\end{aligned} \tag{4.11a}$$

with

$$\left|\tilde{h}(\Omega)\right| \leq 2\Omega^{N+2} \epsilon_h x_0^{-\frac{(N+1)}{2}} \int_1^{\Omega^{-2}u(\delta)} (t-1)^{-1/2}\, t^{\frac{(N+1)}{2}}\, dt \tag{4.11b}$$

At this point, we introduce another small number, Ω_ϵ, on which we impose, first, the condition

$$u(\delta) \geq 2\Omega_\epsilon^2 \tag{4.11c}$$

Expression (4.11a) above can be computed easily for all $\Omega \geq \Omega_\epsilon$. The evaluation of integrals of the type $\int (t-1)^{-1/2} t^\gamma\, dt$ can be done by enclosing the integrand locally in neighborhoods in H^1. We omit the trivial details.

When $\Omega \leq \Omega_\epsilon$, consider first the following trivial Lemma.

Lemma 4.4: *If $R \geq 2$, then*

1. $\displaystyle\int_1^R (t-1)^{-\frac{1}{2}} t^\gamma\, dt \ \leq 2\, R^{\gamma+\frac{1}{2}}$ *when* $\gamma \geq 0$.
2. $\displaystyle\int_1^R (t-1)^{-1/2} t^\gamma\, dt \ \geq 1$ *when* $-1 \leq \gamma$.
3. $\displaystyle\int_1^R (t-1)^{-\frac{1}{2}} t^\gamma\, dt \ = O\Big(R^{\gamma+\frac{1}{2}}\Big)$ *when* $\gamma > -\frac{1}{2}$.

Then, by (4.11c), a., b., c. and Lemma 4.4,

$$\frac{d}{d\Omega}\left(\Omega \int_{r_1(\Omega)}^{\delta} \big(u(r)-\Omega^2\big)^{-1/2}\, \frac{dr}{r}\right)$$

is bounded below by

$$T_1(\Omega) \ \stackrel{\text{def}}{\equiv}\ 4\sqrt{u(\delta)} \sum_{a_n<0} a_n \left(\frac{u(\delta)}{x_0}\right)^{\frac{n}{2}} - \big(u(\delta)-\Omega^2\big)^{-1/2}\, \frac{2u(\delta)}{\delta u'(\delta)} \tag{4.12}$$

where we have set $a_{N+1} = -\epsilon_h$.

Computation of I_3 Here, we also consider two cases: $\Omega \geq \bar{\Omega}_3$ and $\Omega < \bar{\Omega}_3$. The first case is dealt with in a similar manner to I_2. We omit the trivial modifications. The second case is also treated in much the same way, with a few differences coming mainly from the different powers in the asymptotic expansion of u at 0 and at ∞. We include the details, although many of the differences are basically typographical considerations, because conclusions are somewhat different. In particular, as will be noted below, I_3 is mainly responsible for the singularity of F'' at $\Omega = 0$.

Let M be a large number satisfying

a. $u'(M) < 0$ and $u''(x) \geq 0$ on $[M, \infty)$.

b. For a sequence $\{\bar{b}_n\} \in l^1$, we have

$$u(x) = \frac{144}{x^2} u_0(x) = \frac{144}{x^2} \left(1 + \sum_{n=1}^{\infty} \bar{b}_n \bar{x}^{-n\alpha} \right) \qquad x \geq M \tag{4.13a}$$

Furthermore, we know that

$$1 + \sum_{n=1}^{\infty} \bar{b}_n z^n \in \mathcal{U}(I_0, \ldots, I_m; 0, C_g; 2) \qquad I_0 = [1, 1] \tag{4.13b}$$

Here, $\bar{x}$ denotes x/M, and, as a rule, we set $\bar{b}_n = b_n / M^{n\alpha}$.

c. $|u'(x)| \geq 2 \cdot 144\, x^{-3}(1 - \epsilon')$ for $x \geq M$.

Define

$$\bar{t} = \left(\frac{M\, t^{1/2}}{12} \right)^{\alpha}$$

We also consider a small number $\eta \leq u(M)$. It will be chosen so that (4.19) below holds.

We start by obtaining expressions for $u'(r)$ and $u''(r)$ similar to the one for u in (4.13b). Note first that (4.13b) is equivalent to an expression for $y(x)$. Then, by the Thomas–Fermi equation, and by integration, we have

$$y''(x) = \frac{12 \cdot 144}{x^5} \left(\sum_{n=0}^{\infty} \bar{b}_n \bar{x}^{\,-n\alpha} \right)^{3/2} = \frac{12 \cdot 144}{x^5} \sum_{n=0}^{\infty} y''_n \bar{x}^{\,-n\alpha} \qquad y''_0 = 1$$

$$y'(x) = \frac{-3 \cdot 144}{x^4} \sum_{n=0}^{\infty} \left(\frac{4}{4 + n\alpha} \right) y''_n \bar{x}^{\,-n\alpha} = \frac{-3 \cdot 144}{x^4} \sum_{n=0}^{\infty} y'_n \bar{x}^{\,-n\alpha} \qquad y'_0 = 1$$

from which we obtain

$$\begin{aligned} u'(x) = xy'(x) + y(x) &= -\frac{2 \cdot 144}{x^3} \sum_{n=0}^{\infty} \left(\frac{3y'_n - \bar{b}_n}{2} \right) \bar{x}^{\,-n\alpha} \\ &= -\frac{2 \cdot 144}{x^3} u_p(\bar{x}^{\,-\alpha}) \end{aligned} \tag{4.14a}$$

$$\begin{aligned} u''(x) = xy''(x) + 2y'(x) &= \frac{6 \cdot 144}{x^4} \sum_{n=0}^{\infty} (2y''_n - y'_n)\, \bar{x}^{\,-n\alpha} \\ &= \frac{6 \cdot 144}{x^4} u_{pp}(\bar{x}^{\,-\alpha}) \end{aligned} \tag{4.14b}$$

for u_p and u_{pp} in H^1, normalized so $u_p(0) = u_{pp}(0) = 1$.
The strategy will be, as in the case for I_2, to change variables to the inverse function of u, $r(t)$.

Algorithm 4.5: *Given a function*

$$r(t) = 12t^{-1/2} \cdot R(\bar{t})$$

with

$$R(z) \in \mathcal{U}^0(a_0^*, \ldots, a_N^*; C_N, 0; \infty; S) \qquad a_0^* = [1{,}1]$$

satisfying the hypothesis

$$r(t) \geq M \qquad \text{whenever } \bar{t} \in S \subset [-1{,}1]$$

and given

$$g(x) = \sum_{n=0}^{\infty} \bar{c}_n z^n \in \mathcal{U}^1(I_0, \ldots, I_N; C_h, C_g, m) \qquad z = (x/M)^{-\alpha}$$

we compute another neighborhood of type ∞ in C^0, such that, if we set

$$G(\bar{t}) = g\left(r(t)\right) = \sum_{n=0}^{\infty} \bar{c}_n \left(\frac{r(t)}{M}\right)^{-n\alpha}$$

then

$$G(\bar{t}) \in \mathcal{U}^0(d_0^*, \ldots, d_N^*; C, 0; \infty; S)$$

valid for $\bar{t} \in S$.

Description: Consider $a_j \in a_j^*$, for $j = 0, \ldots, N$. Put

$$\left(\sum_{n=0}^{N} a_n \bar{t}^{\,n} + \bar{t}^{N+1} h(\bar{t})\right)^{\gamma} = \sum_{n=0}^{n_0} a_{n,\gamma} \bar{t}^{\,n} + \bar{t}^{\,n_0+1} h(\bar{t}; \gamma, n_0 + 1)$$

with $a_{n,\gamma} \in a_{n,\gamma}^*$, and

$$\left\| h(\bar{t}; \gamma, n_0) \right\|_0 \leq \epsilon_{\gamma,n_0}$$

We can see, on one hand, that

$$\sum_{n=0}^{N}\bar{c}_n\left(\frac{r(t)}{M}\right)^{-n\alpha} = \sum_{n=0}^{N}\bar{c}_n\bar{t}^{\,n}\left(\sum_{k=0}^{N-n}a_{k,-n\alpha}\bar{t}^{\,k}+\bar{t}^{\,N+1-n}h(\bar{t};-n\alpha,N-n+1)\right)$$

$$= \sum_{n=0}^{N}d_n\bar{t}^{\,n}+\bar{t}^{\,N+1}h(t)$$

with

$$|h(t)| \leq \sum_{n=0}^{N}|\bar{c}_n|\,\epsilon_{-n\alpha,N-n+1}$$

$$\leq \sum_{n=0}^{N}|I_n|\,\epsilon_{-n\alpha,N-n+1}+C_g\max_{n=m,\dots,N}\epsilon_{-n\alpha,N-n+1}$$

and

$$d_n = \sum_{i+j=n}\bar{c}_i\cdot a_{j,-i\alpha}$$

$$\in \sum_{i+j=n}I_i\cdot a^*_{j,-i\alpha}\pm\epsilon_n \quad \stackrel{\text{def}}{\equiv} \quad d^*_n$$

where

$$\epsilon_n \leq \begin{cases} C_g\sup_{i=m,\dots,n}|a_{n-i,-i\alpha}| & \text{if } n\geq m \\ 0 & \text{otherwise}\end{cases}$$

On the other hand, since

$$\left|\sum_{n=N+1}^{\infty}\bar{c}_n\left(\frac{r(t)}{M}\right)^{-n\alpha}\right| \leq (C_g+C_h)\left(\frac{r(t)}{M}\right)^{-(N+1)\alpha}$$

$$\leq (C_g+C_h)\,\bar{t}^{\,N+1}\left(1-\sum_{n=1}^{N}|a_n|-C_N\right)^{-(N+1)\alpha}$$

we conclude that

$$g(r(t)) = G(\bar{t}) \in \mathcal{U}^0(d^*_0,\dots,d^*_N;\tilde{C}_h,0;\infty)$$

with

$$\tilde{C}_h = (C_g + C_h)\left(1 - \sum_{n=1}^{N} |a_n| - C_N\right)^{-(N+1)\alpha} + \sum_{n=0}^{N} |I_n|\,\epsilon_{-n\alpha,N-n+1}$$
$$+ C_g \max_{n=m,\dots,N} \epsilon_{-n\alpha,N-n+1}$$

If we cannot check that

$$\left(1 - \sum_{n=1}^{N} |a_n| - C_N\right) > 0$$

the algorithm fails. QED

Now, we analyze $r(t)$.

Algorithm 4.6: *Given $N \geq 0$, we produce intervals $a_1^*, \dots, a_N^*$, and a constant C_N, such that*

$$\left| r(t) - 12\, t^{-1/2}\left(1 + \sum_{n=1}^{N} a_n \bar{t}^{\,n}\right)\right| \leq C_N\, t^{-1/2} \bar{t}^{\,N+1}$$

for constants $a_i \in a_i^*$, $i = 1, \dots, N$, and for $t \leq \eta$.

Description: First, we will construct an inductive procedure to define numbers a_n such that

$$r_N(t) = 12 t^{-1/2}\left(1 + \sum_{n=1}^{N} a_n \bar{t}^{\,n}\right)$$

satisfies

$$u\left(r_N(t)\right) = t\left(1 + O\left(\bar{t}^{\,N+1}\right)\right) \qquad t \to 0 \tag{4.15}$$

By induction. For $N = 0$, let $r_0(t) = 12 t^{-1/2}$. Note that, since $u(x) < 144 x^{-2}$, then $r(t) < r_0(t)$. Also, $r_0(t) \geq M$ for $\bar{t} \leq 1$, and if $t \leq \eta \leq u(M)$ then $\bar{t} \leq 1$. Furthermore, $t \leq \eta \leq u(M)$ implies $r(t) \geq M$, which we will need below.
Therefore, we have

$$u(r_0(t)) = t\left(1 + \sum_{n \geq 1} b_n 12^{-n\alpha} t^{n\frac{\alpha}{2}}\right)$$

Since $u(r(t)) = t$,

$$|r_0(t) - r(t)| \leq t\, \frac{\left|\sum_{n\geq 1} b_n 12^{-n\alpha} t^{n\frac{\alpha}{2}}\right|}{\inf_{r \in [\,r(t)\,,\, r_0(t)\,]} |u'(r)|}$$

Note that hypotheses a. and c. imply

$$|u'(r)| \geq |u'(r_0(t))| \geq \frac{t^{\frac{3}{2}}}{6}(1-\epsilon') \qquad \text{provided } r \in [r(t), r_0(t)] \cap [M, \infty)$$

Our previous remarks, and our assumption on η then imply

$$|r_0(t) - r(t)| \leq C\, t^{\frac{\alpha-1}{2}} \qquad t \leq \eta.$$

Here we have used the fact that $r(t) \geq M$ for $t < \eta$.

For general N, we set

$$r_N(t) = 12\, t^{-1/2} \left(\sum_{n=0}^{N} a_n \bar{t}^{\,n} \right) \qquad a_0 = 1$$

where $a_1, \ldots, a_{N-1}$ satisfy the induction hypothesis.
Note that we have

$$\sum_{n=1}^{\infty} b_n r_N(t)^{-n\alpha} - \sum_{n=1}^{\infty} b_n r_{N-1}(t)^{-n\alpha} = O\big(\bar{t}^{\,N+1}\big) \tag{4.16}$$

Thus, if for any real number γ we put

$$\left(\sum_{n=0}^{N-1} a_n \bar{t}^{\,n} \right)^{\gamma} = \sum_{n=0}^{N} a_{n,\gamma} \bar{t}^{\,n} + O\big(\bar{t}^{\,N+1}\big) \tag{4.17}$$

we see that

$$\begin{aligned} u(r_{N-1}(t)) &= t \left(\sum_{n=0}^{N} a_{n,-2} \bar{t}^{\,n} + O\big(\bar{t}^{\,N+1}\big) \right) \\ &\quad \cdot \left(\sum_{n=0}^{N} \bar{b}_n \bar{t}^{\,n} \Big(\sum_{i=0}^{N} a_{i,-n\alpha} \bar{t}^{\,i} + O\big(\bar{t}^{\,N+1}\big) \Big) + O\big(\bar{t}^{\,N+1}\big) \right) \\ &= t \left(\sum_{n=0}^{N} a_{n,-2} \bar{t}^{\,n} + O\big(\bar{t}^{\,N+1}\big) \right) \cdot \left(\sum_{n=0}^{N} c_n \bar{t}^{\,n} + O\big(\bar{t}^{\,N+1}\big) \right) \end{aligned} \tag{4.18}$$

where

$$c_k = \sum_{n=0}^{k} \bar{b}_n \cdot a_{k-n,-n\alpha} \qquad c_0 = 1$$

By the induction hypothesis, (4.18) is equal to $t(1+O(\bar{t}^{\,N}))$. Thus, using (4.16), we can see that

$$u(r_N(t)) = t\left(\sum_{n=0}^{N} a_{n,-2}\bar{t}^{\,n} - 2a_N\bar{t}^{\,N} + O(\bar{t}^{\,N+1})\right)\cdot\left(\sum_{n=0}^{N} c_n\bar{t}^{\,n} + O(\bar{t}^{\,N+1})\right)$$

for exactly the same c_n as in (4.18).
Therefore, by putting

$$a_N = \tfrac{1}{2}\sum_{m+n=N} a_{n,-2}c_m$$

we get rid of all $\bar{t}^{\,N}$ terms, thus obtaining (4.15).
So far, we have proved the existence of numbers a_n such that (4.15) is satisfied.
This procedure also gives us an algorithm to compute the a_n^*. Indeed, bounds $a_{n,\gamma}^*$ for the $a_{n,\gamma}$ can be computed explicitly, since by the induction hypothesis we already know a_i^*, for $i=1,\dots,N-1$. As for the c_k, recalling (4.13b),

$$c_k = \sum_{n=0}^{k}\bar{b}_n\cdot a_{k-n,-n\alpha} \in \sum_{n=0}^{k} I_n\cdot a_{k-n,-n\alpha}^* \pm \epsilon_k$$

with

$$|\epsilon_k| \le \begin{cases}\sup_{2\le n\le k}|a_{k-n,-n\alpha}|\cdot C_g & \text{if } k\ge 2\\ 0 & \text{if } k\le 1\end{cases}$$

In particular, it is easy to see that $a_1 = \bar{b}_1/2 \in \frac{1}{2}I_1$ (recall (4.13)).

To obtain a good value for the constant C_N, we proceed as follows:
By Algorithm 4.5, we can construct a neighborhood $\mathcal{U}_1^0$ such that

$$f(\bar{t}) = u_0\left(r_N(t)\right) \in \mathcal{U}_1^0(I_0,\dots,I_N;\tilde{\tilde{C}}_h,0;\infty;t\le\eta)$$

(see (4.13a)) provided $r_N(t)\ge M$ for $t\le\eta$. At this point then we check that

$$12\eta^{-\frac{1}{2}}\left(1-\sum_{n=1}^{N}|a_n|\right)\ge M$$

See also (4.19) below.
If we put

$$g(\bar{t}) = \left(\sum_{n=0}^{N} a_n\bar{t}^{\,n}\right)^{-2} \in \mathcal{U}_2(\infty)$$

then,

$$u\left(r_N(t)\right) = t \cdot g(\bar{t}) \cdot f(\bar{t})$$

with

$$F(\bar{t}) = g(\bar{t}) \cdot f(\bar{t}) \in \mathcal{U}^0(\hat{I}_0, \ldots, \hat{I}_N; \hat{C}_h, 0; \infty)$$

and $\mathcal{U}^0$ is the product neighborhood of $\mathcal{U}_1^0$ and $\mathcal{U}_2$. Note that (4.15) implies that $F(\bar{t}) = 1 + O\big(\bar{t}^{\,N+1}\big)$, therefore, we can take $\hat{I}_1 = [1,1]$ and $\hat{I}_i = [0,0]$, for $i = 1, \ldots, N$. This implies that

$$\big|u\big(r_N(t)\big) - t\big| \leq \hat{C}_h \cdot t \cdot \bar{t}^{\,N+1} \qquad t \leq \eta$$

Now, note that

$$|r_N(t) - r(t)| \leq \frac{|u(r_N(t)) - t|}{\inf_{M \leq r \leq \max(r(t), r_N(t))} |u'(r)|}$$

At this point, we check that $a_i < 0$ for all $i = 1, \ldots, N$, which implies that $r_N(t) \leq r_0(t)$. We then conclude, by hypothesis c., that

$$|u'(r)| \geq \big|u'\big(r_0(t)\big)\big| \geq \tfrac{1}{6} t^{3/2}(1 - \epsilon') \qquad M \leq r \leq \max\left(r(t), r_N(t)\right)$$

from which the algorithm follows by taking

$$C_N = \frac{6\hat{C}_h}{(1 - \epsilon')}$$

QED

Now we compute bounds for the derivatives of $r'(t)$, also in the C^0 topology; this will allow us to compute bounds for $w(t) = -\frac{r'(t)}{r(t)}$. We check first that

$$12\eta^{-1/2}\left(1 - \sum_{n=1}^{N} |a_n| - \tilde{C}_N\right) \geq M \qquad \tilde{C}_N = \tfrac{1}{12} C_N \tag{4.19}$$

Algorithm 4.7: *We produce a neighborhood* $\mathcal{U}^0(I_0, \ldots, I_N; C_h, 0; \infty; t \leq \eta)$ *such that*

$$h(t) \stackrel{\text{def}}{\equiv} tw'(t) + w(t) = t^{-1} \cdot f(\bar{t}) \qquad f \in \mathcal{U}$$

for $t \leq \eta$*, where we define* $w(t) = -r'(t)/r(t)$*.*

Description:
We note that
$$r'(t) = \frac{1}{u'\big(r(t)\big)} \qquad r''(t) = -\left(r'(t)\right)^3 u''\big(r(t)\big)$$
As a result of this, in view of (4.14a) and (4.14b) and Algorithm 4.5, we obtain neighborhoods containing functions f, f_p and f_{pp} s.t.
$$r(t) = 12t^{-1/2} f(\bar{t}) \qquad r'(t) = -6t^{-\frac{3}{2}} f_p(\bar{t}) \qquad r''(t) = 9t^{-\frac{5}{2}} f_{pp}(\bar{t})$$
which are valid for $t \le \eta$, and where
$$f_p(\bar{t}) = f^3(\bar{t}) \cdot \left(u_p\Big(\big(\frac{r(t)}{M}\big)^{-\alpha}\Big)\right)^{-1} \qquad f_{pp}(\bar{t}) = f_p^3(\bar{t}) \cdot f^{-4}(\bar{t}) \cdot u_{pp}\left(\big(\frac{r(t)}{M}\big)^{-\alpha}\right)$$
Thus,
$$\begin{aligned} w(t) &= \frac{-r'(t)}{r(t)} = \frac{1}{2t}\left(\frac{f_p(\bar{t})}{f(\bar{t})}\right) \\ w'(t) &= \left(\frac{r'(t)}{r(t)}\right)^2 - \frac{r''(t)}{r(t)} = \frac{1}{4t^2}\left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2 - \frac{3}{4t^2}\left(\frac{f_{pp}(\bar{t})}{f(\bar{t})}\right) \end{aligned}$$
and
$$\begin{aligned} h(t) = tw'(t) + w(t) &= t^{-1}\left(\tfrac{1}{2}\frac{f_p(\bar{t})}{f(\bar{t})} + \tfrac{1}{4}\left(\frac{f_p(\bar{t})}{f(\bar{t})}\right)^2 - \tfrac{3}{4}\frac{f_{pp}(\bar{t})}{f(\bar{t})}\right) \\ &= \frac{1}{4t} f_h(\bar{t}) \end{aligned}$$
for a function f_h belonging to an easily computable neighborhood in C^0. QED

Note now that our choice of functions was normalized so that $f(0) = f_p(0) = f_{pp}(0) = 1$, which implies that $f_h(0) = 0$. This is important: it says that if the Thomas–Fermi potential were *equal* to cx^{-3}, then $F'(\Omega)$ (which still makes sense) would be constant (recall Section 1), and would make Theorem 1.1 completely wrong. But it is not, and one can easily see that
$$f_h(\bar{t}) = -\frac{\alpha^2 b_1}{2 \cdot 12^\alpha} t^{\alpha/2} + O(t^\alpha) \tag{4.20}$$
as follows: recall that
$$r(t) = 12t^{-1/2}\left(1 + \tfrac{\bar{b}_1}{2}\bar{t} + O\big(\bar{t}^{\,2}\big)\right)$$

Note that

$$r'(t) = -6t^{-3/2}\left(1 - \tfrac{(\alpha-1)b_1}{2\cdot 12^\alpha}\, t^{\alpha/2} + O(t^\alpha)\right)$$
$$r''(t) = 9t^{-5/2}\left(1 + \tfrac{(\alpha-1)(\alpha-3)b_1}{6\cdot 12^\alpha} t^{\alpha/2} + O(t^\alpha)\right)$$

which are easily guessed by termwise differentiation of the expression for $r(t)$, and —not so easily— checked using the formulas for $r'(t)$ and $r''(t)$ above. This yields

$$h(t) = \frac{1}{4t}\Bigg(2\big(1 - \tfrac{(\alpha-1)b_1}{2\cdot 12^\alpha} t^{\alpha/2}\big)\big(1 - \tfrac{b_1}{2\cdot 12^\alpha} t^{\alpha/2}\big) + \big(1 - \tfrac{(\alpha-1)b_1}{12^\alpha} t^{\alpha/2}\big)\big(1 - \tfrac{b_1}{12^\alpha} t^{\alpha/2}\big)$$
$$-3\big(1 + \tfrac{(\alpha-1)(\alpha-3)b_1}{6\cdot 12^\alpha} t^{\alpha/2}\big)\big(1 - \tfrac{b_1}{2\cdot 12^\alpha} t^{\alpha/2}\big) + O(t^\alpha)\Bigg)$$

which immediately implies (4.20).

Now, let

$$f_h(\bar t) = \sum_{n=1}^{N} a_n \bar t^{\,n} + H(\bar t)$$

with

$$|H(\bar t)| \le \epsilon_h \, |\bar t|^{N+1} \qquad t \le \eta$$

Finally, then, let L be a large number such that $u(L) \le \eta$: we set $\bar\Omega_3 \le \sqrt{u(L)}$, $b(\Omega) = L$ for all $\Omega \le \bar\Omega_3$, and, arguing as before, we have

$$\begin{aligned}
I_3 &= \frac{d}{d\Omega}\left(\Omega \int_L^{r_2(\Omega)} \big(u(r) - \Omega^2\big)^{-1/2} \frac{dr}{r}\right) \\
&= \frac{d}{d\Omega}\left(\Omega \int_{\Omega^2}^{u(L)} \big(t - \Omega^2\big)^{-1/2} w(t)\, dt\right) \\
&= \frac{d}{d\Omega}\left(\Omega^2 \int_1^{\Omega^{-2}u(L)} (t-1)^{-1/2} w(t\Omega^2)\, dt\right) \\
&= 2\Omega \int_1^{\Omega^{-2}u(L)} (t-1)^{-1/2} h(t\Omega^2)\, dt \\
&\quad - 2\big(u(L) - \Omega^2\big)^{-1/2} w(u(L))u(L) \\
&= \tfrac{1}{2}\Omega^{-1} \sum_{n=1}^{N} \frac{a_n M^{n\alpha}}{12^{n\alpha}} \Omega^{n\alpha} \int_1^{\Omega^{-2}u(L)} (t-1)^{-1/2} t^{\frac{\alpha}{2}n - 1}\, dt \\
&\quad + \tilde h(\Omega) + \big(u(L) - \Omega^2\big)^{-1/2} \frac{2u(L)}{Lu'(L)}
\end{aligned} \qquad (4.21a)$$

with

$$\left|\tilde{h}(\Omega)\right| \leq \tfrac{1}{2}\Omega^{-1+\alpha(N+1)}\epsilon_h \left(\frac{M}{12}\right)^{(N+1)\alpha} \int_1^{\Omega^{-2}u(L)} (t-1)^{-1/2}\, t^{\frac{\alpha}{2}(N+1)-1}\, dt \tag{4.21b}$$

Now, we recall Ω_ϵ, on which we impose now the extra condition

$$u(L) \geq 2\Omega_\epsilon^2 \tag{4.22}$$

Both (4.21a) and (4.21b) can be computed easily for all $\Omega \geq \Omega_\epsilon$. The evaluation of integrals of the type $\int (t-1)^{-1/2} t^\gamma\, dt$ can be done by the same method as in the previous section.

When $\Omega \leq \Omega_\epsilon$, note that the first term in (4.21a) goes to infinity as $\Omega \to 0$, while all the others remain bounded. We use this to obtain a uniform lower bound for the absolute value of this derivative.
By (4.20) we know that $a_1 > 0$; thus, we have

$$\begin{aligned}\frac{d}{d\Omega}\left(\Omega \int_L^{r_2(\Omega)} \left(u(r)-\Omega^2\right)^{-1/2} \frac{dr}{r}\right) \geq \\ \geq \tfrac{1}{2}\Omega^{-1+\alpha} a_1 \left(\frac{M}{12}\right)^\alpha + u(L)^{-1/2} \sum_{\substack{a_n<0\\ n\geq 3}} a_n \left(\frac{u(L)M^2}{144}\right)^{n\frac{\alpha}{2}} \\ + \left(u(L)-\Omega^2\right)^{-1/2} \frac{2u(L)}{Lu'(L)}\end{aligned} \tag{4.23}$$

where we have put $a_{N+1} = -\epsilon_h$.
Note that, since the exponent $\gamma = \alpha - 1$ does not fall under the cases considered in Lemma 4.4, (4.23) is only correct provided $a_2 > 0$. Of course, one can try to modify Lemma 4.4 to include the case $n = 2$, but since it so happens that $a_2 > 0$ there is no need. By this we mean that we check $a_2 > 0$: if the check fails, our proof of Theorem 1.1 fails, and we claim no theorem.

Putting together now (4.1a)—(4.1c), (4.12), (4.11c), (4.22) and (4.23), we conclude that, if

$$\Omega^2 \leq \tfrac{1}{2}\min\bigl(u(L), u(\delta)\bigr)$$

then

$$-F''(\Omega) \geq \tfrac{1}{2}a_1 \left(\frac{M}{12}\right)^\alpha \Omega^{-1+\alpha} + T_1(\Omega) + T_2(\Omega) + T_3 \tag{4.24}$$

for

$$T_2(\Omega) = u(L)^{-1/2} \sum_{a_n<0} a_n \left(\frac{u(L)M^2}{144}\right)^{n\frac{\alpha}{2}} + \left(u(L) - \Omega^2\right)^{-1/2} \frac{2u(L)}{Lu'(L)} < 0$$

$$T_3 = \int_\delta^L \left(y(r)\right)^{-1/2} \frac{dr}{r^{3/2}} > 0$$

The following is a consequence of formulas (4.21a,b), (4.11a,b) and Lemma 4.4.

Proposition 4.8: *We have*

$$F''(\Omega) = -c_0\Omega^{-1+\alpha} + O(1) \qquad c_0 > 0$$

as $\Omega \to 0$.

We now organize the main results in this section in the following algorithm.

Algorithm 4.9: *Given representable δ (small) and L (large), and given neighborhoods in C^0 containing the functions $h(t)$ in Algorithm 4.3 and 4.7, valid for $0 < t \leq u(\delta)$ and $0 < t \leq u(L)$ respectively, we compute strictly positive lower bounds for $-F''(\Omega^*)$ for all thin subintervals Ω^* of Zone I.*

Description: Note that our hypotheses imply that the requirements for the smallness of η and largeness of L have already been checked.
Break up $-F''$ into the three terms in (4.1). I_1 can be computed as described earlier all the way down to $\Omega = 0$.
If $\Omega^* \geq \bar{\Omega}_2$ (similarly for $\bar{\Omega}_3$), I_2 can also be computed as described above. Thus we are left only with the computation of I_2 for $\Omega^* \leq \bar{\Omega}_2$ (similarly for I_3). Note that the dicotomy $\Omega_2^* \geq \bar{\Omega}_2$ or $\Omega_2^* \leq \bar{\Omega}_2$ can be trivially achieved by choosing $\bar{\Omega}_2$ to be one of the endpoints of the Ω^*.
We assume first that $\Omega^* \geq \Omega_\epsilon$. We begin by computing bounds for ${\Omega^*}^{-2}u(\delta)$: if we cannot check that these bounds are greater than or equal to 1, we report a failure and quit. Otherwise, we compute bounds for I_2 using (4.11a) with the error bound for $\tilde{h}$ given by (4.11b). Note that this procedure will prove, in particular, that $\bar{\Omega}_2$ satisfies

the smallness requirement above. This is of mild importance since the choice of $\bar{\Omega}_2$ will not be explicit in our computer implementation.
When $\Omega \leq \Omega_\epsilon$, we simply check that the right hand side of (4.24) is strictly positive for $\Omega = \Omega_\epsilon$. Trivial monotonicity properties will then imply the positivity for $0 < \Omega \leq \Omega_\epsilon$. We also check that requirements (4.11c) and (4.22) for Ω_ϵ are satisfied. QED

5. Zone II.

The purpose of this section is to prove (1.1) for all Ω close to Ω_c.

Lemma 5.1: *Let $f(z)$ be analytic in $|z - z_0| < R$, continuous up to the boundary, with*

1. $f'(z_0) \neq 0$.
2. $f(z) = w_0$ *if and only if* $z = z_0$.
3. *If* $|z - z_0| = R$, *then* $|f(z) - w_0| \geq T$.

Then, there exists $F(w)$ analytic,

$$F : \quad B(w_0, T) \to B(z_0, R)$$

such that $f(F(w)) = w$ for $w \in B(z_0, R)$.

Proof: Consider the curves

$$\Gamma_s(t) = f\big(\gamma_s(t)\big) \qquad |\gamma_s(t) - z_0| = s \quad 0 < s < R$$

with γ_s positively oriented.
Condition 2 implies that

$$n(\Gamma_s, w_0) = \frac{1}{2\pi i} \int_{\gamma_s} \frac{f'(z)}{f(z) - w_0}\, dz$$

is continuous in $0 < s < R$, and it is therefore constant.
Condition 1 says that $n(\Gamma_s, w_0) = 1$ for all s small enough. Thus,

$$n(\Gamma_s, w_0) = 1 \qquad 0 < s < R \tag{5.1}$$

Finally, let $\epsilon > 0$ be given. Condition 3 implies that $B(w_0, T-\epsilon)$ does not intersect Γ_s for $R - \epsilon' < s < R$, for some other ϵ': indeed, assume not: then, there exists z_n, such that $|z_n - z_0| \to R$ such that $|f(z_n) - w_0| < T - \epsilon$. Passing to a subsequence, $z \to z_\infty$ with $|z_\infty - z_0| = R$ and $|f(z_\infty) - w_0| \leq T - \epsilon$, which contradicts 3.
Therefore, $B(w_0, T-\epsilon)$ is contained in one of the connected components of the complement of Γ_s for $R - \epsilon' < s < R$. This implies that the index is constant in w, i.e.

$$n(\Gamma_s, w) = n(\Gamma_s, w_0) = 1 \qquad R - \epsilon' < s < R \qquad w \in B(w_0, T-\epsilon)$$

Thus, if $\alpha_i(w)$ are the solutions of $f(z) = w$ inside $B(z_0, R)$,

$$\sum n(\alpha_i, \gamma_s) = \frac{1}{2\pi i} \int_{\gamma_s} \frac{f'(z)}{f(z) - w}\, dz = n(w, \Gamma_s) = 1$$

for

$$R - \epsilon' < s < R \qquad w \in B(w_0, T-\epsilon)$$

from which, taking $\epsilon \to 0$, we deduce that there is only one α_i and $f'(\alpha_i) \neq 0$. This implies that f^{-1} exists and is analytic. QED

Lemma 5.2: *Let $u \in H^1(|z - r_c| \leq R)$, smooth on the boundary of $B(r_c, R)$, of the form*

$$u(x) = \Omega_c^2 - u_2 R^2\, z^2 + z^3 f(z) \qquad z = \frac{x - r_c}{R} \qquad f(0) = u_3 R^3$$

satisfying

1. *$\|f\| \leq h$, $u_2 > 0$ and $u_2 R^2 > h$.*

2. *For a constant M we have*

$$\left| \frac{d^4}{dx^4} u(x) \right| \leq M \qquad |z| \leq 1$$

Then, $t(x)$ as in (1.4) can be extended analytically to $B(r_c, R)$, and there is an inverse $r(w)$ of $t(x)$, analytic in $|w| < T$ where

$$T \leq \sqrt{u_2 R^2 - h}$$

and

$$\sup_{|w| \leq T} |r'(w)| \leq \frac{2\sqrt{u_2 + hR^{-2}}}{2u_2 - 3|u_3| R - \frac{1}{6} M R^2}$$
$$\left| \frac{d^{n+1} r}{dw^{n+1}}(0) \right| \leq n!\, T^{-n} \, \frac{2\sqrt{u_2 + hR^{-2}}}{2u_2 - 3|u_3| R - \frac{1}{6} M R^2} \qquad n \geq 0$$

Proof: First, note that 1. implies that

$$t(x) = z\sqrt{u_2 R^2 - z f(z)}$$

exists as an analytic function in $x \in B(r_c, R)$ (i.e., $|z| \leq 1$), since the radicand never vanishes and a ball is simply connected, and note also that this definition agrees with (1.4) if x is real.
Also, note that $t(x)$ satisfies the hypothesis of the previous lemma in the circle $x \in B(r_c, R)$. Indeed, $t(x) \neq 0$ unless $x = r_c$ (by 1), and if $|x - r_c| = R$, then $|z| = 1$ and

$$|t(x)| \geq \sqrt{u_2 R^2 - |f(z)|} \geq T$$

Therefore, by the previous lemma, $r(w)$ exists for all $|w| < T$, and we also have $|r(w) - r_c| \leq R$.

Now, note that

$$\sup_{|w|<T} |r'(w)| \leq \sup_{|z|<1} \frac{1}{|t'(x)|} \leq 2 \left(\sup_{|z| \leq 1} \frac{|u(x) - u(r_c)|}{|u'(x)|^2} \right)^{1/2}$$

Since

$$|u(x) - u(r_c)| \leq u_2 |x - r_c|^2 + \frac{h |x - r_c|^3}{R^3}$$

and

$$\begin{aligned} |u'(x)| &\geq 2u_2 |x - r_c| - 3|u_3| |x - r_c|^2 - \tfrac{1}{6} M |x - r_c|^3 \\ &\geq |x - r_c| \left(2u_2 - 3|u_3| R - \tfrac{1}{6} M R^2\right) \end{aligned}$$

we deduce that

$$\frac{|u(x)-u(r_c)|}{|u'(x)|^2} \leq \frac{u_2 + hR^{-2}}{\left(2u_2 - 3\,|u_3|\,R - \frac{1}{6}MR^2\right)^2}$$

The other conclusion follows from Cauchy's inequalities applied to $r'(w)$ on $|w| < T$. QED

We now switch to the notation of Lemma 1.2.

Algorithm 5.3: *Given* $\mathcal{U}_0(I_0, \ldots, I_{2N+1}; C_h, 0; \infty)$ *and bounds* $A^0 < A_0$, *we construct a representable* T *and* $\mathcal{U}_1$, *such that if*

$$u(x) = \Omega_c^2 - z^2 f(z) \qquad z = \frac{x - r_c}{R}$$

and

$$A^0 \leq |y(x)| \leq A_0 \qquad |x - r_c| \leq R$$

with $f \in \mathcal{U}_0$, *then* $w(t) = g(t/T)$, *with* $g \in \mathcal{U}_1$

Description: Note that, since y satisfies (3.3) on $B(r_c, R)$, we have the identities

$$\begin{aligned}
y'''(x) &= \tfrac{3}{2}\frac{y^{1/2}y'}{x^{1/2}} - \tfrac{1}{2}\frac{y^{3/2}}{x^{3/2}} \\
y''''(x) &= \tfrac{3}{4}\left(\frac{y^{-1/2}\,(y')^2}{x^{1/2}} + \frac{2y^2}{x} - \frac{2y^{1/2}y'}{x^{3/2}} + \frac{y^{3/2}}{x^{5/2}}\right) \\
u''''(x) &= 4y'''(x) + x \cdot y''''(x)
\end{aligned}$$

which imply

$$|y''(x)| \leq A_2 \quad \stackrel{\text{def}}{\equiv} \quad A_0^{3/2}/(r_c - R)^{1/2} \tag{5.2a}$$

$$|y'(x)| \leq A_1 \quad \stackrel{\text{def}}{\equiv} \quad \frac{\Omega_c^2}{r_c^2} + A_2 R \tag{5.2b}$$

$$|y'''(x)| \leq A_3 \quad \stackrel{\text{def}}{\equiv} \quad \frac{3A_0^{1/2}}{2(r_c - R)^{1/2}}A_1 + \frac{A_0^{3/2}}{2(r_c - R)^{3/2}} \tag{5.2c}$$

$$|y''''(x)| \leq A_4 \quad \stackrel{\text{def}}{\equiv} \quad \tfrac{3}{4}\left(\frac{{A^0}^{-1/2}A_1^2}{(r_c - R)^{1/2}} + \frac{2A_0^2}{r_c - R} + \frac{2A_0^{1/2}A_1}{(r_c - R)^{3/2}} + \frac{A_0^{3/2}}{(r_c - R)^{5/2}}\right) \tag{5.2d}$$

$$|u''''(x)| \leq M \quad \stackrel{\text{def}}{\equiv} \quad 4A_3 + A_4(r_c + R) \tag{5.2e}$$

Choose representable T and $\tilde{T}$ such that

$$T < \tilde{T} \leq \sqrt{I_0 - h} \qquad h = \sum_{n=1}^{2N+1} |I_n| + C_h$$

and put $\beta = T/\tilde{T}$. Here we assume that $I_0 > h$ and $\beta < 1$. Otherwise, the algorithm fails.

By the previous lemma, $r'(t)$ can be written as $r'(t) = \sum_{n\geq 0} r'_n (t/T)^n$, where r'_n can be computed by power–matching, for $n = 1, \ldots, 2N+1$, because we have $C_g = 0$, and

$$\sum_{n>2N+1} |r'_n| \leq \frac{2R^{-1}\sqrt{I_0 + h}}{R^{-2}\left(2\,|I_0| - 3\,|I_1|\right) - \frac{1}{6}MR^2} \cdot \frac{\beta^{2N+2}}{1-\beta} \tag{5.4}$$

A neighborhood of type ∞ and order $2N+2$ containing $r(t)$ can be obtained by integration.

This allows us to construct a neighborhood $\mathcal{U}_1$ of type ∞ and order $2N+1$ containing the function g in the statement of the algorithm, by simply dividing the neighborhood for r' by the neighborhood for r. QED

Algorithm 5.4: *We compute a bound for F'' in Zone II.*

Description:

Note that

$$\begin{aligned}
\alpha_n &= \frac{1}{\pi} \int_{-1}^{1} (1-t^2)^{-1/2} t^{2n}\, dt \\
&= \frac{1}{2^{2n+1} i^{2n} \pi} \int_0^{2\pi} \left(e^{i\theta} - e^{-i\theta}\right)^{2n} d\theta \\
&= \binom{2n}{n} 2^{-2n}
\end{aligned} \tag{5.5}$$

so their computation poses no difficulty. Note also that $\alpha_n > \alpha_{n+1}$ for all $n \geq 0$.

Therefore, by (1.8), if we set

$$\bar{w}_n = T^n \cdot w_n$$

we can see that

$$-\tfrac{1}{\pi} F'(\Omega) = \Omega \sum_{n=0}^{\infty} \bar{w}_{2n} \left(\frac{D}{T}\right)^{2n} \alpha_n$$

$$= \Omega \sum_{n=0}^{N} \bar{w}_{2n} \left(\frac{D}{T}\right)^{2n} \alpha_n + \Omega \sum_{n>N} \bar{w}_{2n} \left(\frac{D}{T}\right)^{2n} \alpha_n$$

The first term is a polynomial in Ω, so we can easily compute its derivative anywhere. In fact, its derivative equals

$$\sum_{n=0}^{N} \bar{w}_{2n} \alpha_n \gamma^n + \Omega \sum_{n=1}^{N} \bar{w}_{2n} n \alpha_n \gamma^{n-1} \left(\frac{-2\Omega}{T^2}\right)$$

with

$$\gamma = \left(\frac{D}{T}\right)^2$$

Here we check that Zone II is included in the set of Ω that make $\gamma < 1$. Otherwise, we report a failure and we quit the proof.

As for the other term, using Lemma 2.1, and taking

$$C_h \geq \sum_{n>N} |\bar{w}_{2n}|$$

we have

$$\begin{aligned}
&\left| \frac{d}{d\Omega} \left(\Omega \sum_{n>N} \bar{w}_{2n} \left(\frac{D}{T}\right)^{2n} \alpha_n \right) \right| \\
&\qquad \leq C_h \alpha_{N+1} \gamma^{N+1} + 2\Omega_c^2 \sum_{n>N} \frac{\alpha_n |\bar{w}_{2n}| n \gamma^{n-1}}{T^2} \\
&\qquad \leq C_h \alpha_{N+1} \left(\gamma^{N+1} + \frac{2\Omega_c^2}{T^2} \sup_{n>N} n \gamma^{n-1} \right) \\
&\qquad \leq \begin{cases} C_h \alpha_{N+1} \gamma^N \left(\gamma + \dfrac{2(N+1)\Omega_c^2}{T^2} \right) & \text{if } N+1 \geq \frac{1}{|\log \gamma|} \\[2ex] C_h \alpha_{N+1} \left(\gamma^{N+1} + \dfrac{2\Omega_c^2}{eT^2 \gamma |\log \gamma|} \right) & \text{in any case} \end{cases}
\end{aligned}$$

All previous expressions can be easily computed using (5.5). Also, note the slight improvement in the result as a consequence of taking the neighborhoods in the previous algorithm to be of odd order. QED

This concludes the description of all algorithms needed for the proof of Theorem 1.1. We summarize its computer–assisted proof in the following algorithm.

Algorithm 5.5 (Proof of Theorem 1.1): *We produce a constant c such that Theorem 1.1 holds.*

Description: Run Algorithm 3.20 (and algorithms thereof) to obtain all necessary knowledge of the Thomas–Fermi function.

Take $\bar{\Omega}$ as explained at the end of Section 1. to define Zone I and Zone II. As stated earlier in this section, we check that $\gamma(\bar{\Omega}) < 1$. Then, we compute an upper bound for F'' in Zone II, and check that it is strictly negative.

Choose Ω_ϵ, $\bar{\Omega}_2$ and $\bar{\Omega}_3$ as in Section 4, and a partition consisting of fat subintervals of $[\Omega_\epsilon, \bar{\Omega}]$ whose endpoints contain both $\bar{\Omega}_2$ and $\bar{\Omega}_3$ and a subpartition of thin intervals Ω_i^*. We compute the numbers $a_{k,i}$ and $b_{k,i}$. Next, we check that F'' is bounded above by a strictly negative number on the interval $(0, \Omega_\epsilon]$ and on $[\Omega_\epsilon, \bar{\Omega}]$ as described in Algorithm 4.9.

Theorem 1.1 then follows by taking the maximum of all these —finitely many— strictly negative constants. QED

6. Some Extensions.

The purpose of this Section is to extend Theorem 1.1 to a neighborhood of the Thomas–Fermi potential, in an appropriate topology. As pointed out earlier, the fact that Theorem 1.1 holds is a rather delicate one. The following theorem shows this precisely.

Theorem 6.1: *Given any two large numbers N and R, and given ϵ small, there exists a smooth function $f(x)$ such that*

a. $f(x) = y(x)$ for $0 \le x \le R$.

b. For all $x \ge R$, and all $n \ge 0$, we have that

$$\left| \frac{d^n}{dx^n} f(x) - \frac{d^n}{dx^n} y(x) \right| \le \epsilon C_n x^{-3-n}$$

and, however, we also have that $F_f(\Omega)$ vanishes at least N times in $(0,\Omega_c)$. (Note that, if R is large enough, Ω_c is independent of f.)
The C_n are universal constants. In particular, they are independent of ϵ, R and f.

Proof: Note that we can assume R to be as large as we need.

By Corollary 1.3, $F_f''(\Omega)$ is bounded in the range $\Omega_\epsilon \leq \Omega < \Omega_c$. Also, $F_f''(\Omega) = F_y''(\Omega) < 0$ in the same range.
¿From a trivial adaptation of Section 4, it follows that if

$$f(x) = \frac{144}{x^3}\left(1 + bx^{-\alpha}\right) \qquad x \geq R$$

then

$$F_f''(\Omega) = c_1 b\Omega^{-1+\alpha} + O(1) \qquad \alpha = \tfrac{1}{2}(\sqrt{73} - 7) < 1$$

for $c_1 > 0$, uniformly in Ω. Therefore, taking $b = \epsilon$, there is $\Omega_1 \ll \Omega_\epsilon$ such that $F_f''(\Omega_1) > 0$. This gives a function f such that F_f'' has at least one zero. To get more zeros, take R_1 large depending on Ω_1 so that $F_f(\Omega)$, $\Omega \in [\Omega_1, \Omega_c)$, is independent of $f(x)$ outside of $x \in (0, R_1)$. Then, define

$$f(x) = \frac{144}{x^3}\left(1 - \epsilon x^{-\alpha}\right) \qquad x \geq 2R_1$$

and smooth. Then, for Ω_2 small enough $F_f''(\Omega_2) < 0$. This gives us two zeros for F_f''. And so on. QED

¿From this theorem it is then clear that if we want Theorem 1.1 to hold, we need a stronger grip of the behavior of the function f at infinity.

The following theorem is just a consequence of the rest of this article. Its proof is computer assisted.

Theorem 6.2: *There exist N large integer, C and x_1 large constants, and $\epsilon > 0$ and $x_0 > 0$ small, such that if $f(x)$ satisfies*

1. $\|f - y\|_{C^N[x_0, x_1]} \leq \epsilon$.
2. $f(x) = 1 - w\cdot x + x^{3/2} g\left((x/x_0)^{1/2}\right)$ *with* $|w - w_0| \leq \epsilon$, g *analytic and* $\|g - g_0\|_1 \leq \epsilon$, *where* $y_{TF}(x) = 1 - w_0 \cdot x + x^{3/2} g_0\left(x(x/x_0)^{1/2}\right)$.

3. Recall formula (4.13a). *Then,*

$$f(x) = \frac{144}{x^3}\left(1 + \sum_{n=1}^{\infty} \bar{a}_n \bar{x}^{\,-n\alpha}\right) \qquad x \geq R$$

with

$$\sum_{n=1}^{\infty} \left|\bar{b}_n - \bar{a}_n\right| \leq \epsilon$$

Here, ϵ is assumed to be small enough so that our assumptions on f stated at the beginning of Section 1 are satisfied.

Then,

$$F_f(\Omega) \leq c < 0 \qquad \Omega \in (0, \max|rf(r)|)$$

Proof (Computer–Assisted): Take η any small number. If ϵ is small enough, hypothesis 2. and 3. imply that formulas (4.11ab) and (4.21ab) remain valid for $f(x)$ by perturbing the a_n and ϵ_h by at most η percent. Also, for ϵ small enough, hypothesis 1. implies that the value of integral I_1 in (4.1a) remains valid for f also with an error at most η percent. As a consequence, T_1, T_2 and T_3 will change by at most $C\eta$ percent. Therefore,

$$-F_f''(\Omega) \geq \tfrac{1}{2}\tilde{a}_1 \left(\frac{\tilde{M}}{12}\right)^{\alpha} \Omega^{-1+\alpha} - C$$

where $\tilde{a}_1$ and $\tilde{M}$ differ from the ones in (4.20) by at most η percent. In fact, we only need $\tilde{a}_1 > \frac{1}{2}a_1 > 0$.
Therefore, taking $\Omega_0 \ll \Omega_\epsilon$, we see that $F''(\Omega) < c < 0$ for $\Omega \in (0, \Omega_0)$.

Now, set

$$\delta = \inf_{\Omega \in (0, \Omega_c)} \left|F_y''(\Omega)\right| > 0$$

(This is the only point where we use a computer–assisted result.) ¿From formula (1.6), it follows that hypothesis 1., for ϵ small enough, implies that

$$\left|F_f''(\Omega) - F_y''(\Omega)\right| \leq \tfrac{1}{10}\delta \qquad \Omega \in [\Omega_0, \Omega_c)$$

which concludes the proof of the theorem. QED

7. The Implementation

The aim of this section is to provide details about the way algorithms were implemented. The section will be organized as follows:

1. General remarks; in particular, the choice of several heuristic parameters is of special importance for a successful run of the computer proof: we list the approximate values.

2. The second deals with the computer programs, which can divided into two groups.

   a. One is a general package that performs general arithmetic and functional operations on certain general objects. This basic interval arithmetic package is a variation on the one used in [Se1] and [Se2], which in turn is an adaptation of the one developed by D. Rana in [Ra]. It is too long to present here, but we will give enough information about it so that a similar package can be built with little thought. In particular, we will list all function names with a very brief description of each.

      Such packages are quite common already, and probably they will soon be standard.

   b. The other is a package which takes care of the specific functions needed to prove our theorem. It follows very closely the algorithmic presentation in the present paper. We will list all these programs, preceded by a short explanation for each function, which will relate each of them to the corresponding algorithm in the text above.

7.1 General Remarks. According to the general package, (see below) we can store functions locally using the neighborhoods in function space $\mathcal{U}(I_0, \ldots, I_N; C_h, C_g; k)$ introduced in Section 2 as follows: say $f(t) = \tilde{f}(\frac{t-x}{r})$, where $\tilde{f} \in \mathcal{U}$. Then, our knowledge of f can be stored as a structure variable, consisting of

1. A pointer to an array of intervals: it is used to store the I_j.

2. Two integers: one has value N, the order of the Taylor approximation. The other has value k, the type of the neighborhood.

3. Two doubles, to store C_h and C_g.

4. Two doubles, to store x and r.

By considering arrays of the structures above, we can store our *global* knowledge of functions as a single structure variable, consisting of a pointer to an array of the structure variables above. As a consequence, objects like the Thomas–Fermi function $y(x)$, are represented as a single variable. This gives a special computational meaning to Algorithm 3.20, the main result of Section 3.
We divide our remarks according to the section they are related to.

Section 3. In Algorithm 3.20, note that the choice of the x_i and r_i is in principle arbitrary, but in practice, it is very important that they are chosen carefully. Main points to take into account are:

1. All runs of parent algorithms should be successful

2. Error bounds $C_{h,i}$ and $C_{g,i}$ obtained when we run the algorithm are sensitive to our choice of x_i and r_i. The proof Theorem 1.1 is in turn sensitive to these error bounds. In principle, the smaller the r_i the better. It is important that these error bounds are small enough so that we can prove our theorem.

3. The number m is important also: a large m is a consequence of small r_i which will give small error terms for the $C_{\substack{h\\g},i}$, but will make the computation of I_1 in Section 4 very slow, maybe too slow to prove our theorem in a finite time. On the other hand, a small m will speed up the computation of I_1, but will yield bad bounds for the $C_{\substack{h\\g}}, i$. Similar considerations hold for the choice of N.

The choice of N is fixed on a trial and error basis. $N \sim 10$ works. About the x_i and r_i note that their choice was made in Algorithms 3.16 and 3.18. They were picked adaptatively inside the program, in the sense that if during the execution of Algorithm 3.2 (a parent algorithm), one of the error bounds grows outside a pre–specified range, then we make the next r_i a little smaller. And viceversa, if that error goes below a certain range, then we make the next r_i a little bigger. The error we look at in deciding this is $\|p - \tilde{T}p\|/\|p\|$ in Algorithm 3.2, and we wanted it to be within the bounds $[\sim 10^{-17}, \sim 10^{-15}]$. We chose $x_0 \sim 0.008$ and $r_0 \sim 0.0008$. The radii grow as we leave the origin. In carrying out this procedure we made $x_{i+1} \sim x_i + r_i$. This gave enough overlap between intervals to capture the behavior of our functions all over $(x_0 - r_0, x_m + r_m)$. As a result of this method, we obtained $m \sim 800$ and $x_m \sim 300$. The following remark is of mild interest: when choosing the x_i, we instructed the computer to include the point $\sim$2.10 with the idea in mind that r_c is close to this number. Since the information given by Algorithm 3.20 is the only one we would like to use when computing F'', the neighborhoods for $y_{\mathrm{TF}}(x)$ around r_c which would have to be computed in Section 5 turn out a little better.

Section 4. The actual value for $\bar{\Omega}$ is about 0.6956, only $\sim 10^{-3}$ from Ω_c. Thus, Zone II will turn out to be very small. This is unavoidable using our complex–variable methods for Zone II, since with radii larger than that, we cannot exclude the existence of other zeros of $u(z) - \Omega_c^2$ in the vicinity of r_c in the complex plane.

The first heuristic choice we have to make is the numbers a and b as a function of Ω. We describe the choice of a. The choice of b is similar. In the notation of Algorithm 3.20, choose the x_i closest to r_1 such that

$$\sum_{k=1}^{N} \left|I_k^i\right| \leq \delta \cdot \left|I_0^i - \Omega^2\right| \qquad \delta < 1$$

This is a trivial prerequisite if we want to understand $\left(u(r) - \Omega^2\right)^{-3/2}$ in H^1. The choice of δ is delicate, though: if it is very close to 1, then the fractional power operation will yield bad error bounds. If it is very small, we will be forced to take x_i far away from r_1. This will hurt the error terms when we compute I_2, and it could even make the computation of I_2 not possible with our method. The problem is that we will be forced to solve O.D.E.'s at rather large distances (this is already dangerous), and, even worse, we will have to take fractional powers of Taylor expansions with large radii: this may be impossible if, for instance, the solution of the O.D.E. has zeros within our large radius in the complex plane. A value of about 0.3 for δ works most of the time, but it needs fixing for some values of Ω. We refer the reader to functions `alim()` and `blim()` in the program listings for the specific choice of δ as a function of Ω.

We continue now with the computation of I_1, the most time–consuming procedure. The division into fat intervals is done with intervals of length $\sim 10^{-3}$. This is large enough so we can cover the all of Zone I $= [0, \bar{\Omega}]$ with not so many of those fat intervals, around 1000 of them, and it is also small enough so that the approximation given by (4.2) in

terms of the $a_{k,i}$ and $b_{k,i}$ is good enough. As we approach $\bar{\Omega}$, we made the length of these fat intervals smaller, about 10^{-4}. Note that a reduction in the size of the W_i results in having to compute the a_i and b_i more often, but if we are close to Ω_c, the interval $[r_1(\Omega), r_2(\Omega)]$ is very small anyway which require few i, and thin W_i won't hurt. Next, we have to choose $\tilde{a}$ and $\tilde{b}$, or, which is equivalent, we have to choose i_0 and i_1. We chose them to be $i_0 = 10$, $i_1 = n - 10$. This works. The choice of the Ω^* is the most delicate. What we did, is give the computer an initial interval of length l and let it compute bounds for I_1 as well as I_2 and I_3: if these bounds are good enough so that we can show that $-F'' > 0$ on Ω^*, we tell the computer happily to take another Ω^* inside W_i, and do the same, until all of W_i is covered with tests. If for some interval we cannot produce the bound $-F'' > 0$, then we tell the computer to subdivide that interval into two halves, and try each half recursively until (hopefully) we finish. The process finished, so we conclude $-F'' > 0$ all over W_i. The length of the Ω^* that work is about $5 \cdot 10^{-5}$, degenerating until about 10^{-7} near $\bar{\Omega}$: note that this will generate *a lot* of computations.
In principle, we could have given the computer as a first try all of the interval W_i, even without hope, but let the computer figure out how much finer to go before getting the desired bounds. This is fine from the rigorous point of view, but we would be wasting a lot of precious time asking the computer to make checks that we are confident are going to fail. It is thus important to grind W_i into finer intervals before feeding the computer with this recursive procedure.

Concerning the computation of I_2 and I_3, they are analogous, and the only thing worth mentioning is our choice of the following heuristic parameters: $M \sim 291$, $L \sim 295$, $x_0 \sim 0.012$ and $\delta \sim 0.0099$. The degrees of Taylor expansions we chose are 10 for I_3 and 20 for I_2. Also, $\Omega_\epsilon \sim 10^{-2}$, $\bar{\Omega}_2 \sim 0.0932$ and $\bar{\Omega}_3 \sim 0.03469$. See the implementation comments for functions `secder0()` and `secder1()` for more about the $\bar{\Omega}_{2,3}$.

Section 5. We finally discuss the peculiarities of Zone II. Recall that the diameter of Zone II is about 10^{-3}. Also, it follows from Section 5 that it is rather easy to obtain good bounds for the value for $-F''(\Omega_c)$. The analysis for Zone II therefore looks unnecessarily complicated, since it would follow from the apparently easy, but in practice deep statement that $|F''(x)|$ is bounded by about 1000. Since it is numerically evident that $|F''(x)|$ is bounded by a number much smaller than 1000, maybe one can obtain a good bound for F''' which would make the analysis of Zone II trivial.

The only parameter of real importance is R. Too large R are bad because, as mentioned before, it forces us to carry our fractional power analysis to large distances. Too small R will force us to take small $\tilde{T}$ and therefore small T, and will result of restricting our knowledge of $w(t) = g(t/T)$ to a too small neighborhood of Ω_c, thus being unable to cover all of Zone II. Our choice was $R \sim 0.462$. Once R is chosen, we take $\tilde{T}$ as large as we can, still satisfying (5.3), and we are left with the choice of T only. Of course, we would like to take T as large as possible, but note that the closer we take T to $\tilde{T}$, the closer β will get to 1, which will give us a bad error estimate in (5.4). This negative effect can be neutralized by taking N, which until now was arbitrary, to be big, so that the power β^{2N+2} makes the right hand side of (5.4) very small. We took $T \sim 0.0605$ and $N \sim 26$, but larger N will be even better. The only problem with large N is that it will force us to invert a polynomial of large degree. Even with our sloppy implementation of the inversion procedure, errors and speed are of negligible importance.
It follows from these remarks that the analysis of Section 5 will work on an interval around Ω_c whose length depends basically on how large we can take R. Without a more refined analysis, our choice of R is imposed on us by the apparent complex solutions of $u(z) = \Omega_c^2$ around r_c, and thus is not subject to improvement. In other words, there is a good reason for taking $\bar{\Omega} \sim 0.6956$ and not smaller: Zone II is given to us by the problem, not by the computer's ability to compute fast or accurately. As a result, with a slower or less accurate computer, which would not be able to compute $-F''$ in Zone I all the way up to $\bar{\Omega}$, we wouldn't be able to prove our theorem in this way. One would need to perform a real variable analysis to a larger Zone II, in a similar way to the analysis of I_2 and I_3 for $\Omega < \Omega_2, \Omega_3$.

Finally, all programs are written in **C**, and were run on several IBM RS600 simultaneously. As explained later, our problem can be naturally split into several independent processes, making it a very appropriate problem to run on different machines at the same time. Execution took about two days for the programs related to the Thomas–Fermi equation, and about 6 hours for the ones involving the actual computation of F''. Executable files averaged 4Mb each.

7.2 General Purpose Package.

The basic variable types in this package are the following:

```
typedef struct {      double dn;
                      double up;}                 INTERVL;
typedef struct {      double b;}                  BND;
typedef struct {      int deg;
                      INTERVL *p; }               POLY;
typedef struct {      POLY p;
                      BND center;
                      BND r;
                      BND g;
                      int k;
                      BND h;}                     RSERIES;
typedef struct {      int n;
                      RSERIES *f;}                GRS;
union convert{        reps r;
                      unsigned long int i[2];
                      };
```

```
double mtwo = (double) -2;
double mone = (double) -1;
double zero = (double) 0;
double half = (double) 0.5;
double one = (double) 1;
double two = (double) 2;
double eight = (double) 8;
BND bmone = {(double) -1.};
BND b0ero = {(double) 0};
BND bquarter = {(double) .25};
BND bhalf = {(double) .5};
BND bone = {(double) 1};
BND btwo = {(double) 2};
BND bthree = {(double) 3};
BND bfour = {(double) 4};
INTERVL imfour = {(double) -4,(double) -4};
INTERVL imthree = {(double) -3,(double) -3};
INTERVL imtwo = {(double) -2,(double) -2};
INTERVL imone = {(double) -1,(double) -1};
INTERVL izero = {(double) 0,(double) 0};
INTERVL ihalf = {(double) 0.5,(double) 0.5};
INTERVL imhalf = {(double) -0.5,(double) -0.5};
INTERVL ione = {(double) 1,(double) 1};
INTERVL itwo = {(double) 2,(double) 2};
INTERVL ithree = {(double) 3,(double) 3};
INTERVL ifour = {(double) 4,(double) 4};
INTERVL ifive = {(double) 5,(double) 5};
```

```
INTERVL isixteen = {(double) 16,(double) 16};
INTERVL imsix = {(double) -6,(double) -6};
INTERVL ieight = {(double) 8,(double) 8};
INTERVL ifortyeight = {(double) 48,(double) 48};
int n;
double ln2;
INTERVL iln2 = {0.69314718055994484, 0.69314718055994584};
```

The following are the function descriptions.

`up(r)`. Returns a representable strictly larger than `r`.

`dn(r)`. Returns a representable strictly smaller than `r`.

The functions to follow return variable of type `BND`. Variables `a` and `b` are of type `BND`, `x` is of type `INTERVL` and `m` is of type `int`.

`ucvtib(x)`. Returns an upper bound for `x`.

`lcvtib(x)`. Returns a lower bound for `x`.

`cvtdb(d)`. Converts `d` (a `double`) into `BND`.

`cvtintb(m)`. Converts `m` into `BND`.

`uplusb(a,b)`. Returns an upper bound for the sum of `a` and `b`.

`lplusb(a,b)`. Returns a lower bound for the sum of `a` and `b`.

`neg(a)`. Returns $-$`a`.

`absb(a)`. Returns $|$`a`$|$.

`minb(a,b)`. Returns the minimum of `a` and `b`.

`maxb(a,b)`. Returns the maximum of `a` and `b`.

`umultb(a,b)`. Returns an upper bound for the product of `a` and `b`.

`lmultb(a,b)`. Returns a lower bound for the product of `a` and `b`.

`uinvb(a)`. Returns an upper bound for the inverse of `a`.

`linvb(a)`. Returns a lower bound for the inverse of `a`.

`udivb(a,b)`. Returns an upper bound for `a/b`.

`ldivb(a,b)`. Returns a lower bound for `a/b`.

`usquareb(b)`. Returns an upper bound for $\mathtt{b}^2$.

`lsquareb(b)`. Returns a lower bound for $\mathtt{b}^2$.

`upowerb(b,m)`. Returns an upper bound for $\mathtt{b}^m$.

`lpowerb(b,m)`. Returns a lower bound for $\mathtt{b}^m$.

The functions to follow return a variable of type `int`.

`eqb(a,b)`. Returns 1 if `a`=`b`, 0 otherwise.

`neqb(a,b)`. Returns 0 if `a`=`b`, 1 otherwise.

`lsb(a,b)`. Returns 1 if $\mathtt{a} < \mathtt{b}$, 0 otherwise.

`lseqb(a,b)`. Returns 1 if $\mathtt{a} \leq \mathtt{b}$, 0 otherwise.

`grtb(a,b)`. Returns 1 if $\mathtt{a} > \mathtt{b}$, 0 otherwise.

`grteqb(a,b)`. Returns 1 if $\mathtt{a} \geq \mathtt{b}$, 0 otherwise.

The functions to follow return variable of type `INTERVL`, unless stated otherwise. Variables `x` and `y` are of type `INTERVL`, `d` is `double`, `m`, `i` and `j` are of type `int` and `b` is of type `BND`.

`cvtbi(b)`. Converts `b` into `INTERVL`.

`cvtdi(d)`. Converts `d` into `INTERVL`.

`cvtinti(m)`. Converts `m` into `INTERVL`.

`plus(x,y)`. Returns an interval containing the true set–theoretic sum of `x` and `y`.

`neg(x)`. Returns $-\mathtt{x}$.

`iabs(x)`. Returns $|\mathtt{x}|$.

`uabs(x)`. Returns an upper bound to |`x`|. The function returns a variable of type `BND`.

`labsi(x)`. Returns a lower bound to |`x`|. Returns a variable of type `BND`.

`iequ(x,y)`. Returns 1 if the arguments are exactly the same, 0 otherwise.

`ienlarge(x,b)`. Returns an interval containing all points at distance at most `b` from `x`.

`mult(x,y)`. Returns an interval containing the true set–theoretic product of `x` and `y`.

`divi(x,y)`. Returns an interval containing the true set–theoretic division of `x` by `y`. If $0 \in$ `y`, then we abort the program.

`inv(x)`. Returns an interval containing the true set–theoretic inverse of `y`. If $0 \in$ `y`, then we abort the program.

`square(x)`. Returns an interval containing the true set–theoretic square of `x`.

`power(x,m)`. Returns an interval containing the true set–theoretic power $\mathtt{x}^m$.

`intersect(x,y)`. Returns $\mathtt{x} \cap \mathtt{y}$, which also belongs to $\mathcal{I}$.

`iunion(x,y)`. Returns $\mathtt{x} \cup_I \mathtt{y} \in \mathcal{I}$, the smallest interval containing the union of both arguments.

`ration(i,j)`. Returns an interval containing `i/j`.

`iexp(x)`. Returns an interval containing e^x. This can be easily constructed using the Taylor expansion for the exponential.

`ilog(x)`. Returns an interval containing log(`x`). This can be easily constructed using the Taylor expansion for the exponential, in the case $\mathtt{x} \in [\frac{1}{2}, 1)$, and the general case follows trivially after we obtain upper and lower bounds for $\log 2$. These bounds can be obtained heuristically, and then checked using the function `iexp()`. Alternatively, bounds for $\log 2$ are available in the literature, which are better than the ones we could check with `iexp()`; we preferred our way since we simply don't know whether those bounds in the literature are rigorous. This is somewhat wasteful, since `iexp()` is rather conservative (not much). But it did not affect our proof in any noticeable way.

The functions to follow return variables of type `POLY`, unless said otherwise. Arguments starting with `p` are also of type `POLY`, `m` is of type `int`, `x`, `y` and `a` are of type `INTERVL`, and variables starting with `b` are `BND`.

`make_poly(m)`. Returns a `POLY` of degree `m` with zero coefficients.

`polycopy(p)`. Returns a `POLY` identical to `p`.

`coeff(p,m)`. Returns `p.p[m]`, an `INTERVL`.

`coeffmult(p1,p2,m)`. Returns the `m`'th coefficient in the algebraic product (in the interval arithmetic sense) of `p1` and `p2`.

`polysca(p,a)`. Returns bounds for `a`·`p`.

`evalpoly(p,a)`. Returns an `INTERVL` containing the algebraic evaluation of `p` at `a`.

`polynorm(p)`. Returns a `BND`, which is an upper bound for the sum of the absolute value of the coefficients of `p`.

`polyplus(p1,p2)`. Returns bounds for the algebraic sum of the arguments.

`polymult(p1,p2)`. Returns bounds for the algebraic product of the arguments.

`polyscale(p,a)`. Returns bounds for the polynomial in x given by $\mathtt{p}(\mathtt{a} \cdot x)$.

`polyder(p)`. Returns bounds for the algebraic derivative of `p`.

`polyinteg(p)`. Returns bounds for the algebraic integral of `p`.

`polycomp(p1,p2)`. Returns bounds for the algebraic composition `p1(p2)`.

`coeffcomp(p1,p2,m)`. Returns bounds for `m`'th coefficient in the algebraic composition `p1(p2)`.

`polyinv(p)`. Returns bounds for the first `p.deg`+1 Taylor coefficients of the functional inverse p^{-1} such that $\mathtt{p}\circ(p^{-1}) = \mathrm{Id}$.

The following functions return variables of type `RSERIES`, with same radius, center, order and type as the arguments, unless stated otherwise. Variable names continue with the same type, except that those starting with `s` and `r` are now of type `RSERIES`.

`rs(x,r,i)`. Returns a `RSERIES`, with `.center=x`, `.r=r` and `.p.deg = i`. Polynomial coefficients are zero.

`rscopy(r)`. Returns a `RSERIES` identical to `r`.

`ichcoo(a,r)`. Returns bounds for $(\texttt{r.center} - \texttt{a})/\texttt{r.r}$.

`geomrs(x,y,b1,m,b2)`. Returns a neighborhood for $1/(\texttt{x}t+\texttt{y})$ with center at `b1`, radius `b2` and degree `m`.

`rstrunc(s,m)`. Returns a `RSERIES` of degree `m` which contains `s`. It aborts if $\texttt{m} > \texttt{s.p.deg}$.

`rsplusc(r,a)`. Returns bounds for $\texttt{r}+\texttt{a}$.

`rsca(r,a)`. Returns bounds for $\texttt{a}\cdot\texttt{r}$.

`rsplus(r,s)`. Returns bounds for the sum of the arguments. The order and type are the smaller of those of the arguments. It is assumed without check that the center of the arguments are identical.

`rsminus(r,s)`. Returns bounds for $\texttt{r}-\texttt{s}$. The order and type are the smaller of those of the arguments. It is assumed without check that the center of the arguments are identical.

`rsmult(r,s)`. Returns bounds for the product of the arguments. The order and type are the smaller of those of the arguments. It is assumed without check that the center of the arguments are identical.

`rseval(r,a)`. Returns an `INTERVL` with bounds for $\texttt{r}(\texttt{a})$.

`rsinteg(r)`. Returns a neighborhood for the functions $\int_{\text{r.center}}^{x} f(t)\, dt$ where $f \in \texttt{r}$, and $|x - \texttt{r.center}| < \texttt{r.r}$. The polynomial order and type of the output is one more than those of the argument.

`rsdint(r,a,b)`. Returns a `INTERVL` with bounds for $\int_b^a f(t)\, dt$ where $f \in \texttt{r}$.

`rsoverx(s)`. Returns bounds for $\texttt{s}(x)/(x-\texttt{s.center})$. It is assumed here without check that $\texttt{s}(\texttt{s.center})=0$ exactly and the type of `s` is at least 1.

`rslog(x,y,b1,m,b2)`. Returns a neighborhood for $\log(\texttt{x}t+\texttt{y})$ with center at `b1`, radius `b2` and degree `m`.

`ievders(s,x)`. Returns an `INTERVL` containing bounds for the derivative of `s` at `x`.

ievnders(s,x,m). Returns an INTERVL containing bounds for the m'th derivative of s at x.

bl1nrs(s). Returns a BND which contains an upper bound for $\|s\|_1$.

rstimesx(s). Returns bounds for the functions $x \cdot \mathtt{s}(x)$.

frac22(x,b). Returns a BND with an upper bound for $C_{2.2}(x,b)$, as in Section 2.

frak22(x,b). Returns a BND with an upper bound for $K_{2.2}(x,b)$.

rsmatpower(s,*r). Returns the argument *r, a pointer to an array of RSERIES, containing bounds for all powers of s, from 0 to s.p.deg.

rspower(r,x). Returns bounds for $\mathtt{r}^x$.

polypower(p,x). Returns a POLY containing bounds for the Taylor approximation of degree p.deg of $\mathtt{p}^x$.

The functions to follow perform operations on variables of type GRS, represented by arguments starting with gs.

Functions

```
grscopy(gs)
grsmult(gs1,gs2)
grseval(gs,x)
grsdereval(gs,x)
grstimesx(gs)
grspower(grs,x)
```

perform the corresponding operations as their RSERIES counterparts on each RSERIES member of their structures.

grsdint(gs,x,y). Returns bounds for the integral from x to y of all global functions in gs. Note here that the role the of x and y is reversed with respect to rsdint().

grs(n). This function simply returns a GRS with .n member equal to n, and with space allocated for n+1 variables of type RSERIES. Note that the further allocation needed *in* the POLY member of RSERIES is not done here. This should be done using either rs() or make_poly() above.

`grsintpt(gs,i)`. Returns a `double`, a heuristic choice for the "middle" point between the centers of the `i`'th and `i`+1'th members of `gs`. If we denote the centers by x_1 and x_2, and corresponding radii by r_1 and r_2, this function returns approximately the number $\frac{r_1 \cdot x_2 + r_2 \cdot x_1}{r_1 + r_2}$.

`grsloc(gs,x)`. Another heuristic function. Returns an integer representing the member of the structure `gs` which best captures the behavior of `gs` near `x`, i.e., the one that minimizes (in a heuristic way), the output of `ichcoo(x,...)` above.

The functions with names equal to the above followed by an `f` perform the same operations, plus: they destroy the arguments containing pointers by freeing the memory they have allocated.

In addition to these functions, we also have the following, which are of an entirely heuristic nature. They are designed to make the heuristic guesses of p in Algorithms 3.2 and similar. They manipulate polynomials, this time defined simply as arrays of `SIZE`+1 variables of type `double` (we took `SIZE`=50 in our programs); we will denote such variables here with names starting with `po`. They also use the additional `extern` variable `DEGREE`, smaller than `SIZE` at all times, intended to allow us to vary the effective degree of these polynomials inside the programs. They do not return any variables a values, only as arguments. All operations they perform are floating–point.

`pzer(po)`. Initializes `po` to 0.

`pcopy(po1,po2)`. Copies `po1` into `po2`.

`pnorm(po1)`. Returns a `double` with a floating–point approximation to $\|$`po1`$\|_1$.

`psub(po1,po2,po3)`. Puts in `po3` the algebraic difference of `po1` and `po2`.

`pprod(po1,po2,po3)`. Puts in `po3` the algebraic product of `po1` and `po2` truncated to `DEGREE`.

`pinte(p01,po2)`. Puts in `po2` the algebraic integral of `po1`, truncated to `DEGREE`.

`psca(po1,x,po2)`. Puts in `po2` the product of `po1` by the `double x`.

`pscale(po1,x,po2)`. Puts in `po2` the scaled polynomial `po1`(`x` $\cdot\, t$).

`myprpower(po1,x,po2)`. Takes `po1` to the power `x` and puts the result in `po2`.

Last, but not least, we also need functions that give the *decimal* expansion of rationals bounds for representable numbers. This is required, for instance, to be able to state Lemma 3.21 in the form we did, rather than in a form where the bounds claimed are given in the harder to visualize hexadecimal form. The construction of such functions, while not trivial, is not too hard and we omit the details.

7.3 Aperiodicity Programs. The following is a brief itemized explanation of the computer programs included at the end of this paper.

Throughout the programs, we will use the external variables

```
extern GRS Y, YRS, U;
extern INTERVL Le, UL, De, UD, C1, W, RC, BC, ALPHA;
extern RSERIES HINF, H0;
```

The variable `ALPHA` will contain consist of bounds for $\frac{1}{2}(\sqrt{73}-7)$ computed once and for all at the beginning of each program. The variables `De` and `UD` correspond to δ and $u(\delta)$ of Section 4, and `Le` and `UL` correspond to variables L and $u(L)$ also in Section 4, and they are introduced in functions `h_at_0()` and `hinf()` respectively. The rest will be explained below.

`lipreg()`. Implements Algorithm 3.1, returning the Lipschitz norm if we can show that it is less than 1, and returning 1 otherwise.

`vtffx()`. Implements Algorithm 3.3. Algorithm 3.2, which is needed for the execution of the former, is implemented explicitly inside the function.

`supervtffx()`. Implements Algorithm 3.5. Note that in this function, as well as in `vtffx()`, the values and derivatives of the solution of the ODE are returned as arguments, while the `double` that the function returns as value is an approximation to $\|p-\tilde{T}p\|$, which, as pointed out before, will be used in deciding how much to increase or decrease the next choice of r_i.

`vtffxi()`. Implements Algorithm 3.2. Gives neighborhoods of type 2. This function (and `vtffxi2()` below) returns a neighborhood valid for all centers in the interval `xin`. As a consequence, no `.center` of type `BND` can be naturally specified in the `RSERIES` it returns; we assigned the value 0 (a better choice would be `NaN`). This means that we cannot manipulate the outcome of this particular function with any general–purpose function which would attempt to make use of the structure member `.center`. All such manipulations should be done explicitly taking into account that the centers are contained in `xin`.[1]

`tfypoly()`. See below.

`vtffxi2()`. Implements Algorithm 3.2. Gives neighborhoods of type ∞. The power matching scheme is done in `tfypoly()`

`lip0()`. Implements Algorithm 3.6, returning the Lipschitz norm if we can show that it is less than 1, and returning 1 otherwise.

`vtff0()`. Implements Algorithm 3.8. Again, Algorithm 3.7 is built in.

`y_at_0()`. Implements Algorithm 3.7.

`tfw()`. Implements Algorithm 3.15. In our description of this Algorithm above, the x_i and r_i are given. From the logical point of view, this is true, but from the computational point of view, the x_i and r_i are produced within `tfw()`.

The successive rigorous bounds we find for w_0 are printed in hexadecimal form as they are obtained. The reason for this is that it takes a long time to run each iteration. In this way, one has rigorous bounds for w_0 even if the function does not finish (due to computer shut down, or impatience on our part).

`tff(w)`. This function implements Algorithm 3.16, where the `INTERVL w` has as endpoints a rigorous upper and lower bound for w_0. As in `tfw()`, the choice of the x_i and r_i is done inside the function. The values y_i and y'_i are returned as a single variable of type `GRS`.

`tfrs(y)`. Implements Algorithm 3.20. The argument `y`, which is of type `GRS`, is the output of `tff(w)`.

`Omega(u)`. The argument `u`, a `GRS` variable, contains bounds for $u_{\mathrm{TF}}(x) = x \cdot y_{\mathrm{TF}}(x)$.

---

[1] One could attempt to define a new variable type in which centers are of type `INTERVL`. Since this is the only place in which we use this special choice, we decided not to do it in this particular situation.

The function then returns an interval $[r^{\mathrm{dn}}, r^{\mathrm{dn}}]$ which is guaranteed to contain r_c. Recall that r_c is uniquely defined by the identity $u'(r_c) = 0$. Thus, we first look in a heuristic manner for r^{dn} and r^{up}, and conclude that they are valid bounds after checking that $u'(r^{\mathrm{dn}}) \geq 0$ and $u'(r^{\mathrm{up}}) \leq 0$, which we can easily do using the general purpose interval arithmetic package, namely, function `grsdereval()`.

The heuristic construction of the interval is done via a bisection method, slightly modified so that the interval produced is optimal, in the sense that any representable $r > r^{\mathrm{dn}}$, the bounds we obtain for $u'(r)$ would not be strictly positive (similarly for r^{up}).

`tfprint()`. This is a bookeeping function. It prints the output of `tfw()`, bounds for w_0 and $-b_1$, the output of `tff()` (Algorithm 3.16), `tfrs()` (Algorithm 3.18), together with a `GRS` variable containing u_{TF}, the bounds for r_c produced in `Omega()`, and bounds for Ω_c^2. The bounds for u_{TF}, and Ω_c^2 are easily obtained via the general purpose functions `grstimesx()` and `grseval()`.

The print out is done in hexadecimal form, so that it can be printed on a file and read rigorously for later use.

`tfread()`. This function simply reads the output of `tfprint()`.

`r0(w)`. Given an `INTERVL` `w`, this function produces an interval $[a, b]$ containing all solutions $r \leq r_c$ to the equation $u(r) = w^2$, for all values w contained in `w`. The bounds are, first, obtained heuristically using bisection (as in `Omega()`), and seen to be correct by checking that an interval containing $u(b)$ is entirely to the right of (i.e. larger than or equal to) $\mathtt{w}^2$, and an interval containing $u(a)$ is entirely to the left of $\mathtt{w}^2$.

`r1(w)`. Same as before, except that the solutions we are looking for are $u(r) = w^2$ for $r \geq r_c$. Both functions produce optimal intervals, in the sense described in `Omega()`. This function only returns a true bound when $\mathtt{w} \geq 10^{-20}$. This is perfectly fine, since it is only invoked for $\mathtt{w} \geq \Omega_\epsilon$, and we check that $\Omega_\epsilon > 10^{-20}$ (if fact, $\Omega_\epsilon \sim 0.01$).

`vtfinf()`. Implements Algorithm 3.13, for representable values of b=`-a0` and R=`t`. As usual, Algorithm 3.12 is implemented inside.

`tfc1()`. We use our bounds for w_0, which `tff()` transforms for bounds for y_{TF}, and, for a representable `ctest` we return 1 if we can guarantee that $b_1 \leq -$`ctest`, -1 if $b_1 \geq -$`ctest` and 0 if we cannot guarantee any inequality.

`getc1()`. Organizes `tfc1()` to implement Algorithm 3.19. The bounds for $-b_1$ are stored in the external variable `C1`. As in `tfw()`, instead of returning an interval value for our bounds at the end, this function prints the successive rigorous bounds it obtains in hexadecimal form.

`refiney()`. This function implements Algorithm 3.18, with the extra obvious feature that instead of obtaining y_i^* and ${y'}_i^*$ alone, it takes care of comparing them with the old bounds we had, given by function `tff()` and stored in `Y`, and takes the intersection of them. This requires the x_i to be the same as before, which poses no problem, of course.

`refine_numbers()`. Similar to `tfprint()`, except that, once bounds for b_1 are computed, it takes care of using them to improve the bounds for y_{TF} before printing them out.

`tfw2(w)`. According to Algorithm 3.19, once new bounds for b_1 are obtained, and the corresponding bounds for y_{TF} are obtained, one can attempt to improve the bounds for w_0. This function takes care of this, by returning 1 if, using the scheme in Algorithm 3.19, we can show that $w_0 \geq$ `w`, -1 if $w_0 \leq$ `w`, and 0 if we cannot claim any inequality.

`mygetw()`. This function simply organizes the previous one.

`refineY()`. This function implements Algorithm 3.16 again. The difference with `tff()` is only a programming one, since the bounds this function computes are assumed to be refinements of previous ones. Thus, the x_i need not be recomputed.

`rstfu2(x,r,y)`. This function implements a trivial variant of Algorithm 3.2, in the type ∞ case, except that instead of returning bounds for y_{TF} alone (which are returned in the pointer variable `y`), it returns $r \cdot y_{\mathrm{TF}}(r)$ also. As pointed out before, multiplication by r, which is implemented as a general purpose routine `rstimesx()`, is not available here, since `vtffxi2()` returns a `RSERIES` without a meaningful `.center` structure member, needed in `rstimesx()`.

`yinf(t,m)`. This function implements Algorithm 3.12. The `int` variable `m` represents the order of the expansion we want.

`expandY()`. Given the variable `Y` of type `RSERIES` containing bounds for y_{TF} and y'_{TF} at certains points x_i, this function returns another `RSERIES` with the same bounds at the same x_i, plus the trivial bounds $y_{\mathrm{TF}} \in [0,1]$, $y'_{\mathrm{TF}} \in [-2,0]$ at other points x_i, chosen heuristically inside it. This is justified since if at the stage we use this function we already know that $w_0 \leq 2$, which we do because by the time we use this function we would have already run `mygetw()`. Obviously, this bound can be replaced by any other we know to be true by the time we run this function, and it will probably have no effect on the final answer, since the information `expandY()` produces will most probably never be used until `refineY()` has already improved it to a quite sharp bound. After doing this, it also destroys `Y` by freeing the memory allocated to it. Note that this function has a purely administrative role.

The functions below refer to the algorithms presented in Section 4. In the explanation to follow, we use the notation introduced there.

`secder0(w,a)`. Computes I_2 in Section 4, for the thin interval `w` and $a =$ `a`. If `w` $\leq \bar{\Omega}_2$, it simply invokes `secder0_sp()` below. Note that $\bar{\Omega}_2$ is implicitly defined by the first `if` statement in this function.

`secder1(w,a)`. Same as before, but for I_3 this time.

`dermatrix()`. Computes the numbers $a_{k,i}$ and $b_{k,i}$ involved in the computation of I_1 in Section 4. They are stored in the polynomial part of `RSERIES` variables.

`secdermat(w,a1,a2,der1,der2,i)`. Uses the numbers $a_{k,i}$ and $b_{k,i}$ (given in `a1`, `a2`, `der1` and `der2` respectively) to compute $\tilde{J}_i($`w`$)$. The choice of the t_i is made using `grsintpt()`.

`secderdir()`. This function computes J_i directly, i.e. computes bounds for the functions f_i in Section 4 involved in the computation of I_1, without use of the numbers $a_{k,i}$ and $b_{k,i}$. This function is intended to compute these f_i for representable arguments.

`super_secderdir()`. This function does the same as the previous, but for interval values of the argument. In other words, for the thin interval under consideration $[z_1, z_2]$, this function uses the previous one to compute the $f_i(z_1)$ and $f_i(z_2)$, and then sets $J_i = f_i(z_1) \cup_I f_i(z_2)$. Recall that this is justified due to the monotonicity of the f_i.

`secder_help()`. This function uses the previous one to compute bounds for I_1. It selects the i_0 and i_1, computes the $\tilde{J}_i$ either directly or using the $a_{k,i}$ and $b_{k,i}$, and adds them up together.

`alim()`. This function selects heuristically the number a (as a function of Ω) involved in the break up of I into the I_i ($i = 1, 2, 3$) in (4.1).

`blim()`. Same as before, but for b.

`secder2()`. This function organizes the previous ones to produce bounds for $-F''$ in a thin interval Ω.

`supersecder2()`. Given an interval Ω, it runs the previous function to check whether $-F''$ is strictly positive. If we can check that it is, it reports a success and returns the bounds. If it is not, it subdivides the interval and tries each half recursively. Note that the fact that this function eventually finishes implies that $-F''$ is strictly positive on the original interval.

`supersupersecderdn(w,vup,dup)`. (At this stage, the reader will probably notice our lack of imagination in picking names for all the functions involved in this proof.) Given a fat interval `w`, and numbers $a_{2,i}$ (stored in `vup`), and $b_{2,i}$ (stored in `dup`), corresponding to the upper endpoint of `w`, this function computes the missing $a_{1,i}$ and $b_{1,i}$ corresponding to the lower endpoint of `w`; then, it breaks `w` into thin subintervals using the heuristic variable `step`, and then invokes the previous function to check that $-F''$ is strictly positive in each thin subinterval. Once this is done, we know that $-F''$ is strictly positive all over `w`. Before returning, this function replaces the arguments `vup` and `dup` by the values of the $a_{1,i}$ and $b_{1,i}$ corresponding to the lower end of `w`. The reason for this will be explained in the next function.

`secderdn(r,step)`. Given representable `r` and `step`, this function computes the a_i and b_i corresponding to r, constructs the fat interval $w_1 = [\texttt{r} - \texttt{step}, \texttt{r}]$, and gives them to the previous function (note that these are exactly the arguments it needs) to do its job. When `supersupersecderdn()` is finished, it returns to us the a_i and b_i corresponding to the lower end of w_1; then, we construct the new fat interval $w_2 = [\texttt{r} - 2 \cdot \texttt{step}, \texttt{r} - \texttt{step}]$, and we give it to `supersupersecderdn()` again. Note that the a_i and b_i that we need now are exactly the ones returned to us by `supersupersecderdn()`. And so on. Note that in the construction of the w_i we used expressions of the type $\texttt{r} - i \cdot \texttt{step}$; there is no need to make these computations rigorous, as long we make sure that the lower endpoint of each interval is exactly the upper endpoint of the next, which is trivial to arrange.

We do not include any stopping criterion for this function, rather, we instruct it to print in exact hexadecimal form each fat interval on which we can guarantee that $-F''$ is strictly positive. The reason for this is that it takes a very long time to do each fat interval; thus, we prefer to let several computers run (say six of them) on complementary ranges, and stop them as they redundantly start to get into each other's territory.

`supersupersecderup()`. Same as `supersupersecderdn()`, but designed to go up, rather than down.

`secderup()`. "Up" version of `secderdn()`.

`hinf()`. Implements Algorithm 4.7. All sub–algorithms are explicitly implemented inside as needed. `Le` is chosen here.

`h_at_0()`. Implements Algorithm 4.3. Also, sub–algorithms are implemented inside. `De` is chosen inside also.

`printh()`. Runs the two previous functions, and types the output in hexadecimal form for later use.

`tfint1(alpha,x)`. This function computes bounds for

$$\int_1^{1+x} (t-1)^{-1/2} t^\alpha \, dt$$

for $x < 1$. It does it by Taylor–expanding the integrand around 1.

`tfint2(alpha,a,b)`. Computes rough bounds for

$$\int_a^b (t-1)^{-1/2} t^\alpha \, dt$$

by bounding t^α and integrating $(t-1)^{-1/2}$.

`tfint3(alpha,a,b)`. Computes rough bounds for

$$\int_a^b (t-1)^{-1/2} t^\alpha \, dt$$

by bounding $(t-1)^{-1/2}$ and integrating t^α.

`tfint4(alpha,a,b)`. Computes precise bounds for

$$\int_a^b (t-1)^{-1/2} t^\alpha \, dt$$

by Taylor–expanding the integrand. This function is to be used when we require precision and are willing to give up speed. The two previous ones are intended for a fast, rather inaccurate answer.

`tfintf1(alpha,x)`. This function computes rough bounds for

$$\int_1^{1+x} (t-1)^{-1/2} t^\alpha \, dt$$

for all `x`. It does it by using `tfint1()` around 1, and using `tfint2()` (or `tfint3()`) in several other small intervals away from 0.

`tfintf2(alpha,x)`. As the previous function, but using `tfint4()` instead for precise, slow bounds.

`secder0_sp(w)`. Computes I_2 when $\Omega_\epsilon \le \mathtt{w} \le \bar{\Omega}_2$, as explained in Section 4. It checks that $\mathtt{w}^2 \le u(\delta)$: otherwise, it aborts the program. Thus, if the program eventually ends without abortions, we are guaranteed that $\bar{\Omega}_2^2 \le u(\delta)$.

`secder1_sp()`. Computes I_3 when $\Omega_\epsilon \le \Omega \le \bar{\Omega}_3$. The same comments as `secder0_sp()` apply.

`secder0_speps()`. Computes $T_1(\Omega)$, as in Section 4.

`secder1_speps()`. Computes $T_2(\Omega)$, as in Section 4. As before, it also checks that $\mathtt{w}^2 \le u(L)$: otherwise, it aborts the program. Thus, if the program eventually ends without abortions, we are guaranteed that $\bar{\Omega}_3^2 \le u(L)$.

`secder_eps()`. Computes T_3.

The next functions are related to Section 5. We also use the same notation used there.

`rtpoly()`. See below.

`tfwz(t,x)`. Implements Algorithm 5.3. The neighborhood $\mathcal{U}_0$ is computed using `vtf-fxi2()`, where $R = \mathtt{x}$. The value `t` is T in the statement of the algorithm, which we check it is less than or equal to $\tilde{T}$. The power–matching scheme is done in `rtpoly()`.

`sdinv(w0,w)`. As in Algorithm 5.4, computes bounds for $-F''$ in the interval `w0`, using `w`, the output of `tfwz()`.

`super_sdinv(a,ww)`. Given an interval `a` (which will be all of Zone II), and `ww`, the output of `tfwz()`, this function subdivides `a` recursively until it checks, using the previous function, that $-F''$ is strictly positive in each subinterval of `a`. When this function exits, we know that $-F''$ is strictly positive all over `a`.

In order to display how the previous functions can be used to prove Theorem 1.1, we conclude the present discussion with a list of the final programs used in proving our theorem. In doing this, we omit the trivial, but lengthy, statements such as those dealing with variable declarations.

The following obtains —from scratch— bounds for w_0, which are printed in exact hexadecimal form.

```
W.dn = (double) 1;
W.up = (double) 2;
tfw(W,0.0);
```

Note that it looks as if in the previous program we are assuming the apparently trivial bounds $[1,2]$ for w_0 before we start. In fact, we are not, since these initial bounds are used only to make heuristic choices where to look. The only thing to bear in mind, is that the choices we will be using will be in $[1,2]$. Therefore, once we exit the program, we only have to check that there is at least one of those choices for which we were able to conclude that it bounds w_0 from above, and that there is one of those choices that bounds w_0 from below. Once we know this, the final bounds obtained will be true bounds for w_0.

Using the bounds for w_0 obtained before, we can now obtain bounds for y_{TF} and use these to obtain bounds for b_1.

```
W=readivalio();
printivalio(W);
Y=tff(W);
Y=expandY();
printgrsio(Y);
fflush(stdout);
C1.up = (double) 14;
C1.dn = (double) 13;

getc1();
```

In the previous program, note that without the statement `expandY()`, the points at which we have bounds for `Y` may not be very many, and when trying to solve the ODE backwards, starting at the largest x_i stored inside `Y`, we may run into trouble, since, as pointed out in Section 3, we need large x_i to be able to solve the ODE around ∞. Note also that the bounds stored in `Y` are printed out in exact hexadecimal form, since we will be using them in all programs to follow.

Next, we organize the output of the previous program so that it contains, first, the bounds for w_0, next, the bounds for $-b_1$, and then the bounds for y_{TF} contained in `Y`, all in exact hexadecimal form. Then, this output can be used as input for the following program, which will use the new bounds for $-b_1$ to obtain improved bounds for y_{TF}, stored in `Y`. It will also compute the corresponding `YRS` and `U`.

```
W=readivalio();
C1 = readivalio();
Y=readgrsio();

refine_numbers();
```

The next program refines our bounds for w_0, b_1 and y_{TF}, as described in Algorithm 3.19. The output is printed out with same format as usual in exact hexadecimal form.

```
tfread();
mygetw();
refineY();
getc1();

refine_numbers();
```

A comment concerning the previous program. Since the bounds that the previous procedures yield are quite sharp, the computer may have to solve ODE's with initial values close to the critical ones that cause the solutions to vanish, but only very slowly. As a result, when trying to check bounds for w_0 with `mygetw()`, some choices of `wtest` may yield a failure of some of the ODE–solving algorithms, which will cause the previous program to be aborted. The thing to do in this case is to take whatever bounds were successfully obtained by `mygetw()`, use them to replace the old bounds for w_0 written in some file, and restart the previous program without using `mygetw()`. To achieve even greater accuracy, one may also rerun the previous program (after replacing the bounds for w_0 with the new ones) with a different choice of the heuristic parameter `t` in `mygetw()`. These comments extend also to b_1 and `getc1()`, although those occurrences are very unlikely in this case.

Once we are happy with all the bounds for the Thomas–Fermi data, we run the following program, which will write in exact hexadecimal form the neighborhoods for $h(t)$ in Algorithm 4.7, for $h(t)$ in Algorithm 4.3, and bounds for L, $u(L)$, δ and $u(\delta)$.

```
tfread();

printh();
```

The program to follow will check that $-F''$ is strictly positive for $\Omega_\epsilon \leq \Omega \leq \Omega_c$.

```
tfread();
readh();
i1 = ratpower(BC,1,2);
i1.dn = 0.6956; /*This amounts to setting Zone II= [0.6956,Ωc]*/
res = tfwz(0.0605,0.462);
super_sdinv(i1,res);
/*At this point we know that -F'' > 0 on Zone II*/
secderdn(i1.dn,1.e-4);
```

This program can be complemented with programs of the type

```
tfread();
readh();
secderup(x,t);
```

or

```
tfread();
readh();
secderdn(x,t);
```

for `double` values of `x` and `t`, which can run on separate computers.

Note that these three last programs will tell us in exact hexadecimal form which fat intervals W are guaranteed to satisfy $-F'' > 0$ on W, but will never stop trying to get W closer and closer to 0. The thing to do is, as long as we see that we have checked all intervals inside $[\sim 10^{-2}, \Omega_c]$, halt the program, set Ω_ϵ equal to the lower end of the last interval checked, and run the following last program:

```
tfread();
readh();
i1 = readivalio();
b1 = minb(uabs(UL),uabs(UD));
b1 = ldivb(b1,btwo);
printbd(b1);
if(lsb(b1,ucvtib(square(i1)))){
printf("error");
abort();
}
i3 = secder1_speps(i1);
i4 = secder0_speps(i1);
i2 = secder_eps();
i2 = plus(i2,neg(plus(i3,i4)));
```

```
i3 = poweri(i1,plus(imone,ALPHA));
i3 = mult(i3,divi(HINF.p.p[1],itwo));
i3 = mult(i3, poweri(divi(cvtbi(HINF.center),cvtinti(12)),ALPHA));
i2 = plus(i2,i3);
if(lseqb(lcvtib(i2),b0ero)){
printf("error");
abort();
}
else printf("PROVED");
```

This program checks that $T_1(\Omega) + T_2(\Omega) + T_3 > 0$ for all $\Omega \in$ `i1`, after checking that $\Omega_\epsilon =$ `i1.up` satisfies (4.11c) and (4.22). As a result, any $\Omega_\epsilon \in$ `i1` would finish the proof. In our case, `i1.dn`=`i1.up`=lower end of the last thin interval for which we successfully run `supersupersecderdn()`.

References

[Ar] Arnold, V. "*Mathematical Methods of Classical Mechanics* " Springer.

[EKW] Eckmann, J. P., Koch, H. and Wittwer, P. "*A computer Assisted Proof of Universality in Area Preserving Maps*" Memoirs, A.M.S., Vol 289 (1984).

[EW] Eckmann, J. P. and Wittwer, P. "*Computer Methods and Borel Summability Applied to Feigenbaum's equation*" Lecture Notes in Mathematics **227**, Springer Verlag (1985).

[FL] Fefferman, C. and Llave, R., "*Relativistic Stability of Matter, I*", Revista Matemática Iberoamericana Vol 2 no.1&2, pp. 119-213 (1986)

[FS1] Fefferman, C. and Seco, L. "*The Ground–State Energy of a Large Atom* " Bull. A.M.S., Vol **23** no. 2, 525—530, 1990

[FS2] Fefferman, C. and Seco, L. "*Eigenvalues and Eigenfunctions of Ordinary Differential Operators* " To appear in *Adv. Math.*

[FS3] Fefferman, C. and Seco, L. "*The Eigenvalue Sum for a One–Dimensional Potential* " To appear in *Adv. Math.*

[FS4] Fefferman, C. and Seco, L. "*The Density in a One-Dimensional Potential* " To appear in *Adv. Math.*

[FS5] Fefferman, C. and Seco, L. "*The Eigenvalue Sum for a Three–Dimensional Radial Potential* " To appear in *Adv. Math.*

[FS6] Fefferman, C. and Seco, L. "*The Density in a Three–Dimensional Radial Potential* " To appear in *Adv. Math.*

[FS7] Fefferman, C. and Seco, L. "*On the Dirac and Schwinger Corrections to the Ground–State Energy of an Atom* " To appear in *Adv. Math.*

[KM] Kaucher, E. W. and Miranker, W. L., "*Self-validating Numerics for Function Space Problems*", Academic Press, New York (1984).

[HKSW] Helffer, B., Knauf, A., Siedentop, H., Weikard, R. "*On the Absence of a First–Order Correction for the Number of Bound States of a Schrödinger Operator with Coulomb Singularity* " To appear in "*Comm. P.D.E.*"

[Hi] Hille, E., "*On the Thomas–Fermi Equation* " Proc. Nat. Acad. Sci, USA, 62, 7—10.

[LL] Lanford, O. and Llave, R. "*Solution of the Functional Equation for Critical Circle Mappings with Golden Rotation Number*" *In preparation.*

[Ll] Llave, R. "*Computer Assisted Bounds in Stability of Matter*" Computer Aided Proofs in Analysis, IMA Series in Math. and Appl. Vol **28**. Cincinnati (1989). Springer.

[Lo] Lohner, R., "*Einschließung der Lösung gewöhnlicher Anfangs– und Randwertaufgaben und Anwendungen* " Dissertation, Universität Karlsruhe (TH), 1988.

[Mo] Moore, R. E., "*Methods and Applications of Interval Analysis*", S.I.A.M., Philadelphia (1979).

[Ra] Rana , D., "*Proof of Accurate Upper and Lower Bounds for Stability Domains in Denominator Problems*", Thesis, Princeton University (1987)

[Se1] Seco, L. "*Lower Bounds for the Ground State Energy of Atoms* " Thesis, Princeton University, 1989.

[Se2] Seco, L., "*Computer Assisted Lower Bounds for Atomic Energies*" Computer Aided Proofs in Analysis, IMA Series in Math. and Appl. Vol **28**, 241—251. Cincinnati (1989). Springer.

[SW2] Siedentop, H., Weikard, R. "*On the Leading Correction of the Thomas–Fermi Model: Lower Bound*" and an appendix by A.M.K. Müller. Inv. Math., Vol., 97, pp 159—193, 1989.

```
#include <math.h>
#include <stdio.h>
#include "extern.inc"
INTERVL r0(), r1();
RSERIES yinf(), y_at_0(), vtffxi();



INTERVL
secder0(w,a)
INTERVL w, a;
{
        INTERVL yw0, yw1, x, ith;
        INTERVL secder0_sp(), coo, coo12, sol;
        BND b;
        RSERIES y, u, y2;
        int i;

        coo = U.f[1].p.p[0];
        if(lsb(ucvtib(square(w)),lcvtib(coo)))return(secder0_sp(w));

        ith = divi(cvtinti(-3),cvtinti(2));
        x = r0(w);
        yw0 = grseval(YRS,x);
        yw1 = grsdereval(YRS,x);
        y = vtffxi( x, a, yw0, yw1);

        u = rscopy(y);
        for(i = y.p.deg; i>=1; --i)
                u.p.p[i] = plus(mult(y.p.p[i],x),mult(y.p.p[i-1], cvtbi(y.r)));
        u.p.p[0] = mult(y.p.p[0] ,x);
        u.g = umultb(y.g,uplusb(ucvtib(x),y.r));
        u.h = umultb(y.h,uplusb(ucvtib(x),y.r));
        u.h = uplusb(u.h,umultb(uabs(y.p.p[y.p.deg]),y.r));
        u.p.p[1] = intersect(u.p.p[1],mult(grsdereval(U,x),cvtbi(u.r)));

        y2 = rs(y.p.deg-1,y.center,y.r);
        for(i=0; i <= y2.p.deg; ++i) y2.p.p[i] = u.p.p[i+1];
        y2.g = u.g, y2.h = u.h;
```

secder0

```
	y2 = rspowerf(y2,divi(ith,itwo));
	y2 = rsmultf(y2,y2);
	y2 = rsmultf(y2,y);

	sol = izero;
	coo = divi(iabs(plus(a,neg(x))),cvtdi(y.r.b));
	if(coo.up > 1) coo.up = (double) 1;
	coo12 = iexp(divi(ilog(coo),imtwo));
	if(coo12.dn < 1) coo12.dn = (double) 1;
	for(i = y2.p.deg; i >= 0; i-- ){
		sol = mult(sol,coo);
		sol = plus(sol,divi(y2.p.p[i],plus(cvtinti(i),neg(ihalf))));
	}


	sol = mult(sol, coo12);
	b =  uplusb(umultb(y2.g,btwo),udivb(y2.h,
	    lplusb(cvtintb(y2.p.deg+1),negb(bhalf))));
	sol = mult(ienlarge(sol,b), cvtdi(y2.r.b));


	freep(u.p), freep(y2.p);
	return(sol);
}

INTERVL
secder1(w,a)
INTERVL w, a;
{
	INTERVL yw0, yw1, x, ith;
	INTERVL secder1_sp(), coo, coo12, sol;
	BND b;
	RSERIES y, u, y2;
	int i;

	coo = U.f[U.n-1].p.p[0];
	if(lsb(ucvtib(square(w)),lcvtib(coo)))return(secder1_sp(w));
```

secder1

```
ith = divi(cvtinti(−3),cvtinti(2));
x = r1(w);
yw0 = grseval(YRS,x);
yw1 = grsdereval(YRS,x);
y = vtffxi( x, a, yw0, yw1);

u = rscopy(y);
for(i = y.p.deg; i>=1; −−i)
        u.p.p[i] = plus(mult(y.p.p[i],x),mult(y.p.p[i−1], cvtbi(y.r)));
u.p.p[0] = mult(y.p.p[0] ,x);
u.g = umultb(y.g,uplusb(ucvtib(x),y.r));
u.h = umultb(y.h,uplusb(ucvtib(x),y.r));
u.h = uplusb(u.h,umultb(uabs(y.p.p[y.p.deg]),y.r));
u.p.p[1] = intersect(u.p.p[1],mult(grsdereval(U,x),cvtbi(u.r)));




y2 = rs(y.p.deg−1,y.center,y.r);
for(i=0; i <= y2.p.deg; ++i) y2.p.p[i] = neg(u.p.p[i+1]);
y2.g = u.g, y2.h = u.h;

y2 = rspowerf(y2,divi(ith, itwo));
y2 = rsmultf(y2, y2);




y2 = rsmultf(y2,y);
sol = izero;
coo = divi(iabs(plus(a,neg(x))),cvtdi(y.r.b));
if(coo.up > 1) coo.up = (double) 1;
coo12 = iexp(divi(ilog(coo),imtwo));
if(coo12.dn < 1) coo12.dn = (double) 1;
for(i = y2.p.deg; i >= 0; i−− ){
        sol = mult(sol,coo);
        if(i % 2 )
                sol = plus(sol,divi(y2.p.p[i],plus(cvtinti(−i),ihalf)));
        else
```

```
                sol = plus(sol,divi(y2.p.p[i],plus(cvtinti(i),neg(ihalf))));
        }


        sol = mult(sol, coo12);
        b =  uplusb(umultb(y2.g,btwo),udivb(y2.h,
            lplusb(cvtintb(y2.p.deg+1),negb(bhalf))));
        sol = mult(ienlarge(sol,b), cvtdi(y2.r.b));

        freep(u.p), freep(y2.p);
        return(sol);
}
```

dermatrix

```
dermatrix(w, sec, thi)
INTERVL w;
RSERIES *sec, *thi;
{
        INTERVL mw2, ifh;
        INTERVL  t1, t2;
        RSERIES rsw1, rsw;
        int i, i0, i1;

        ifh = divi(cvtinti(-5),cvtinti(2));
        mw2 = neg(square(w));

        i0 = grsloc(U,r0(w));
        i1 = grsloc(U,r1(w));

        i = i0+5;
        t1 = cvtdi(grsintpt(U,i));
        while(i <= i1-5){
                ++i;
                if(i== 60 *(i/60 ))
                        printf("%d points done\n", i);
                fflush(stdout);

                rsw = rscopy(U.f[i]);
```

```
                rsw.p.p[0] = plus(rsw.p.p[0],mw2);
                t2 = cvtdi(grsintpt(U,i));
                rsw1 = rspower(rsw,divi(ifh,ifour));
                rsw1 = rsmultf(rsw1,rsw1);
                rsw1 = rsmultf(rsmult(rsw1,YRS.f[i]),rsw1);
                thi->p.p[i] = mult(mult(ithree,w),rsdint(rsw1,t2,t1));
                rsw1 = rsmultf(rsw1,rsw);
                sec->p.p[i] = rsdintf(rsw1,t2,t1);
                t1 = t2;
        }


}
```

secdermat

```
INTERVL
secdermat(w, a1, a2, der1, der2, i)
INTERVL w;
RSERIES a1, a2;
RSERIES der1, der2;
int i;
{
        INTERVL i1, v1, v2, de1, de2, sol;

        v1 = a1.p.p[i];
        v2 = a2.p.p[i];
        de1 = der1.p.p[i];
        de2 = der2.p.p[i];

        i1 = plus(v1,neg(v2));
        i1 = divi(i1,plus(cvtbi(a1.center),neg(cvtbi(a2.center))));
        i1 = mult(i1,plus(w,neg(cvtbi(a1.center))));
        i1 = plus(i1,v1);
        sol.up = i1.up;

        i1 = mult(de1,plus(w,neg(cvtbi(a1.center))));
        i1 = plus(i1,v1);
        sol.dn = i1.dn;
```

```
        i1 = mult(de2,plus(w,neg(cvtbi(a2.center))));
        i1 = plus(i1,v2);
        sol.dn = maxm(sol.dn,i1.dn);
        if(sol.up < sol.dn){
                printf("SECDERMAT: negative interval !!!\n");
                printival(sol);
                printf("w:");
                printival(w);
                printf("\ncenter1:");
                printbd(a1.center);
                printf("v1:");
                printival(v1);
                printf("d1:");
                printival(de1);
                printf("\ncenter2:");
                printbd(a2.center);
                printf("v2:");
                printival(v2);
                printf("d2:");
                printival(de2);
                fflush(stdout);
                abort();
        }

        return(sol);
}
```

```
INTERVL
secderdir(mw2, t1, t2, i)
INTERVL  mw2, t1, t2;
int i;
{
        RSERIES rsw;

        rsw = rscopy(U.f[i]);
        rsw.p.p[0] = plus(rsw.p.p[0],mw2);
        rsw = rspowerf(rsw,ration(-3,4));
        rsw = rsmultf(rsw,rsw);
```

secderdir

```
        rsw = rsmultf(rsw,rscopy(YRS.f[i]));
        return(rsdintf(rsw,t2,t1));
}
```

super_secderdir

```
INTERVL
super_secderdir(mw2, t1,t2,i)
INTERVL t1, t2;
INTERVL  mw2;
int i;
{
        return(iunion(secderdir(cvtdi(mw2.up),t1,t2,i),
            secderdir(cvtdi(mw2.dn),t1,t2,i)));
}
```

secder_help

```
INTERVL
secder_help(w, a, b, v1, v2, de1, de2)
RSERIES v1, v2, de1, de2;
INTERVL w, a, b;
{
        INTERVL mw2;
        INTERVL sol, t1, t2;
        int i, i0, i1;

        mw2 = neg(square(w));

        sol = izero;
        i0 = grsloc(U,a);
        if(U.f[i0].center.b < a.dn)++i0;
        i1 = grsloc(U,b);
        if(U.f[i1].center.b > b.up)--i1;

        t1 = a;
        for(i= i0; i <= i1-1; ++i){
                t2 = cvtdi(grsintpt(U,i));
                if(
                        i == i0 ||
                    v1.p.p[i].up == (double) 0 ||
                    v2.p.p[i].up == (double) 0
```

```
                ){
                        sol = plus(sol, super_secderdir(mw2,t1, t2,i));
                }
                else sol = plus(sol,secdermat(w,v1, v2, de1, de2,i));
                t1 = t2;
        }
        sol = plus(sol, super_secderdir(mw2,t1, b,i));

        return(sol);
}
```

alim

```
double
alim(w)
INTERVL w;
{
        int j, i;
        double rat, diff;
        RSERIES rsw;
        INTERVL x, w2;


        rat = 3.0;
        if(w.dn > 0.136 && w.up < .140) rat = 2.0;
        x = U.f[1].p.p[0];
        if(lsb(ucvtib(square(w)),lcvtib(x))){
                if(De.up == De.dn) return(De.up);
                else printf("De error\n");
                fflush(stdout);
                abort();
        }

        w2 = square(w);

        i = 0;
        while(U.f[i].p.p[0].dn < w2.up){
                ++i;
        }
```

```
        diff = 1.0;
        --i;





        while(diff > 0.0){
                ++i;
                rsw = U.f[i];
                diff = (w2.dn - rsw.p.p[0].dn)/rat;
                for(j=1; j <= rsw.p.deg; ++j)
                        diff += fabs(rsw.p.p[j].up);
        }
        --i;






        return(grsintpt(U,i));
}


double
blim(w)
INTERVL w;
{
        int j, i;
        double rat, diff;
        RSERIES rsw;
        INTERVL w2;


        w2 = U.f[U.n-1].p.p[0];
        if(lsb(ucvtib(square(w)),lcvtib(w2))){
                if(Le.up == Le.dn) return(Le.up);
                else printf("Le error\n");
                fflush(stdout);
                abort();
```

blim

```
	}


	if(w.up < .09){
		w2 = r1(w);
		return(w2.dn-16.0);
	}

	w2 = square(w);

	i = U.n;
	while(U.f[i].p.p[0].dn < w2.up)--i;

	diff = 1.0;
	++i;
	rat = 2.1;
	if(w.dn > 0.695) rat = 1.5;
	if(w.up < 0.2388 && w.dn > .1) rat = 1.8;
	if(w.up < 0.1668 && w.dn > .1) rat = 1.6;
	/*else if(w.dn > 0.136 && w.up < .140) rat = 1.8;*/
	while(diff > 0.0){
		--i;
		rsw = U.f[i];
		diff = (w2.dn - rsw.p.p[0].dn)/rat;
		for(j=1; j <= rsw.p.deg; ++j)
			diff += fabs(rsw.p.p[j].up);
	}



	return(grsintpt(U,i));
}
```

secder2

```
INTERVL
secder2(w, v1, v2, de1, de2)
RSERIES v1, v2, de1, de2;
INTERVL w;
{
```

```
        INTERVL sol3, sol1, sol2;
        double i1, i2;

        i1 = alim(w);
        i2 = blim(w);

        sol1 = secder1(w,cvtdi(i2));

        sol2 = secder_help(w,cvtdi(i1),cvtdi(i2), v1, v2, de1, de2);
        sol3 = secder0(w,cvtdi(i1));
        return(plus(plus(sol1,sol2),sol3));
}
```

supersecder2

```
INTERVL
supersecder2(w, v1, v2, de1, de2)
RSERIES v1, v2, de1, de2;
INTERVL w;
{
        INTERVL sol, w1, w2;
        sol = secder2(w, v1, v2, de1, de2);
        if(sol.dn <= (double) 0){
                w1.up = w.up;
                w1.dn = 0.5*(w.up+w.dn);
                w2.up = w1.dn;
                w2.dn = w.dn;
                return(iunion(supersecder2(w1, v1, v2, de1, de2),
                    supersecder2(w2, v1, v2, de1, de2)));
        }
        return(sol);
}
```

supersupersecderdn

```
INTERVL
supersupersecderdn(w, vup, dup)
INTERVL w;
RSERIES *vup, *dup;
{
        INTERVL w1, sol;
        RSERIES v2, de2;
        double step = 5.e-5;
```

```
        if(w.dn > 0.672)step = 8.e-6;
        if(w.dn > 0.688)step = 3.e-6;
        if(w.dn > 0.693)step = 1.5e-6;
        if(w.dn > 0.694)step = 4.e-7;
        if(w.dn > 0.6956)step = 2.e-7;

        v2 = rs(U.n,cvtdb(w.dn),b0ero);
        de2 = rs(U.n,cvtdb(w.dn),b0ero);
        dermatrix(cvtdi(w.dn),&v2, &de2);
        w1.up = w.up;
        w1.dn = w1.up - step;
        sol = supersecder2(w1, *vup, v2, *dup, de2);
        w1.up = w1.dn;
        w1.dn -= step;
        while(w1.dn > w.dn + step/5.0){
                sol = iunion(sol,supersecder2(w1, *vup, v2, *dup, de2));
                w1.up = w1.dn;
                w1.dn -= step;
        }
        w1.dn = w.dn;
        sol = iunion(sol, supersecder2(w1, *vup, v2, *dup, de2));
        freep(vup->p), freep(dup->p);
        *vup = v2;
        *dup = de2;
        return(sol);
}
```

secderdn

```
secderdn(r, step)
double r, step;
{
        RSERIES v, de;
        INTERVL w;

        v = rs(U.n,cvtdb(r),b0ero);
        de = rs(U.n,cvtdb(r),b0ero);
        dermatrix(cvtdi(r),&v, &de);
        w.dn = r;
```

```
	while(w.dn > 0){
		w.up = w.dn;
		w.dn -= step;
		printival(supersupersecderdn(w, &v, &de));
	}
}
```

supersupersecderup

```
INTERVL
supersupersecderup(w, vup, dup)
INTERVL w;
RSERIES *vup, *dup;
{
	INTERVL w1, sol;
	RSERIES v2, de2;
	double step = 0.00005;

	if(w.dn > 0.672)step = 8.e-6;
	if(w.dn > 0.688)step = 3.e-6;
	if(w.dn > 0.693)step = 1.5e-6;
	if(w.dn > 0.694)step = 4.e-7;
	if(w.dn > 0.6956)step = 2.e-7;

	v2 = rs(U.n,cvtdb(w.up),b0ero);
	de2 = rs(U.n,cvtdb(w.up),b0ero);
	dermatrix(cvtdi(w.up),&v2, &de2);
	w1.dn = w.dn;
	w1.up = w1.dn + step;
	sol = supersecder2(w1, *vup, v2, *dup, de2);
	w1.dn = w1.up;
	w1.up += step;
	while(w1.up < w.up - step/5.0){
		sol = iunion(sol,supersecder2(w1, *vup, v2, *dup, de2));
		w1.dn = w1.up;
		w1.up += step;
	}
	w1.up = w.up;
	sol = iunion(sol, supersecder2(w1, *vup, v2, *dup, de2));
	freep(vup->p), freep(dup->p);
	*vup = v2;
```

```
        *dup = de2;
        return(sol);
}
```

secderup

```
secderup(r, step)
double r, step;
{
        RSERIES v, de;
        INTERVL w;

        v = rs(U.n,cvtdb(r),b0ero);
        de = rs(U.n,cvtdb(r),b0ero);
        dermatrix(cvtdi(r),&v, &de);
        w.up = r;
        while(w.up > 0){
                w.dn = w.up;
                w.up += step;
                printival(supersupersecderup(w, &v, &de));
        }
}
```

rtpoly

```
POLY
rtpoly(u)
POLY u;
{
        POLY res, res2;
        int i;

        res = make_poly(u.deg-2);
        for(i=0; i <= res.deg; ++i) res.p[i] = neg(u.p[i+2]);
        res = polypowerf(res,ihalf);
        res2 = make_poly(res.deg+1);
        for(i=0; i <= res.deg; ++i) res2.p[i+1] = res.p[i];
        freep(res);
        return(polyinv(res2));
}
```

tfwz

```
RSERIES
tfwz(t, x)
```

```
double t, x;
{
        INTERVL i2, a0l, a4, i1, rmr, a0, a1, a2, a3;
        BND m, tt, h;
        RSERIES u, y, rstfu2(), r, rp;
        int oldeg;

        /**** t = 0.0605, and x = 0.462 will do the job for w=0.6956 ****/


        oldeg = DEGREE;
        DEGREE = SIZE -1;
        r.k = rp.k = 0;
        r.h = b0ero;
        r.g = b0ero;
        rp.h = b0ero;
        rp.g = b0ero;

        u = rstfu2(RC,x, &y);
        rmr = plus(RC,cvtbi(negb(u.r)));
        a0l = ienlarge(y.p.p[0],bl1nrsf(rsplusc(y,neg(y.p.p[0]))));
        a0 = cvtbi(bl1nrs(y));
        a2 = divi(ratpower(a0,3,2),ratpower(rmr,1,2));
        a1 = plus(divi(BC,square(RC)),mult(a2,cvtbi(u.r)));

        a3 = mult(a1,mult(ration(3,2),ratpower(a0,1,2)));
        a3 = plus(a3,divi(ratpower(a0,3,2),mult(itwo,rmr)));
        a3 = divi(a3,ratpower(rmr,1,2));



        a4 = divi(ratpower(a0,3,2),rmr);
        a4 = plus(a4, mult(mult(itwo,a1),poweri(a0,ihalf)));
        a4 = divi(a4,rmr);
        a4 = plus(a4,divi(square(a1),poweri(a0l,ihalf)));
        a4 = divi(a4,poweri(rmr,ihalf));
        a4 = plus(a4, mult(itwo,divi(square(a0),rmr)));
        a4 = mult(a4, ration(3,4));
```

```
m = uabs(plus(mult(ifour,a3),mult(a4,plus(RC,cvtbi(u.r)))));

r.p = rtpoly(polyscale(u.p,inv(cvtbi(u.r))));
r.r = cvtdb(t);

tt = labsi(u.p.p[2]);
i1 = neg(divi(u.p.p[2],square(cvtbi(u.r))));
i2 = u.p.p[2];
u.p.p[0] = u.p.p[1] = u.p.p[2] = izero;
h = bl1nrs(u);
tt = lplusb(tt,negb(h));
if(lseqb(tt,b0ero)){
        printf("TFWZ: BAD !!!\n");
        fflush(stdout);
        abort();
}
tt =  lcvtib(ratpower(cvtbi(tt),1,2));

h = udivb(h,lmultb(u.r,u.r));
rp.h = ucvtib(poweri(cvtbi(uplusb(uabs(i1),h)),ihalf));
i2 = plus(iabs(mult(i2,itwo)),neg(iabs(mult(ithree,u.p.p[3]))));
i2 = divi(i2,square(cvtbi(u.r)));
i2 = ienlarge(i2,umultb(udivb(m,cvtintb(6)),upowerb(u.r,2)));
rp.h = umultb(btwo,udivb(rp.h,lcvtib(i2)));

if(lseqb(rp.h,b0ero)){
        printf("TFWZ: radius in the expansion for U is too large !!!\n");
        fflush(stdout);
        abort();
}

r.h = udivb(cvtdb(t),tt);
if(lseqb(bone,r.h)){
        printf("TFWZ: too large t !!!\n");
        fflush(stdout);
        abort();
}
r.h = udivb(upowerb(r.h,r.p.deg),lplusb(bone,negb(r.h)));

rp.h = umultb(rp.h,r.h);
```

```
        r.h = udivb(rp.h,cvtintb(r.p.deg+1));
        r.h = umultb(r.h, cvtdb(t));

        rp.p = polyder(r.p);
        rp.p = polyscalef(rp.p,cvtbi(r.r));
        r.p = polyscalef(r.p,cvtbi(r.r));
        r.p.p[0] = RC;
        rp.r = r.r;
        r.center = b0ero;
        rp.center = b0ero;

        r = rsmultf(rp,rspowerf(r,imone));
        DEGREE = oldeg;

        return(r);
}
```



```
INTERVL
sdinv(w0, w)
INTERVL w0;
RSERIES w;
{
        int i;
        INTERVL  a[30], sol2, sol, g;
        BND b;
        double t;
        int m;


        t = w.r.b;
        a[0] = ione;
        a[1] = ihalf;
        for(i=0; i <= 29; ++i)
                a[i] = divi(ichoose(cvtinti(2*i),i),power(cvtinti(2),2*i));

        g = plus(BC,neg(square(w0)));
        g = divi(g,square(cvtdi(t)));
```

```
        if(grteqb(ucvtib(g),bone)){
                printf("SDINV: Omega value tries to escape domain of convergence !!!. \n");
        }

        if(w.p.deg % 2) m = (w.p.deg-1)/2;
        else m = (w.p.deg)/2;

        sol  = mult(w.p.p[2*m],a[m]);
        for(i=m-1; i >= 0; --i)
                sol = plus(mult(sol,g),mult(w.p.p[2*i],
                    a[i]));

        sol2 = mult(mult(cvtinti(m),w.p.p[2*m]),a[m]);
        for(i=m-1; i >= 1; --i)
                sol2 = plus(mult(sol2,g),
                    mult(mult(cvtinti(i),w.p.p[2*i]),a[i]));

        sol = plus(sol,mult(sol2,neg(mult(itwo,square(divi(w0,cvtbi(w.r)))))));

        b = uabs(inv(ilog(g)));
        if(grteqb(cvtintb(m+1),b)){
                b = umultb(cvtintb(2*(m+1)),ucvtib(divi(BC,square(cvtbi(w.r)))));
                b = umultb(uplusb(uabs(g),b),
                    upowerb(uabs(g),m));
        }
        else{
                b = umultb(btwo,ucvtib(divi(BC,square(cvtbi(w.r)))));
                b = udivb(b,labsi(mult(mult(iexp(ione),g),ilog(g))));
                b = uplusb(upowerb(uabs(g),m+1),b);
        }

        b = umultb(w.h, umultb(uabs(a[m+1]), b));
        return(ienlarge(sol,b));
}
```

super_sdinv

```
super_sdinv(a, ww)
INTERVL a;
RSERIES ww;
```

```
{
        INTERVL sol;
        RSERIES  w;

        w = rstrunc(ww,53);
        sol = sdinv(a, w);
        if(sol.dn > (double) 0){
                printf("SUCCESS for ");
                printival(a);
                printivalio(a);
                fflush(stdout);
        }

        else {
                printf("trying...\n");
                fflush(stdout);
                sol.up = a.up;
                sol.dn = .5*(a.dn+a.up);
                super_sdinv(sol,w);
                a.up = sol.dn;
                super_sdinv(a,w);
        }
}


RSERIES
hinf()
{
        RSERIES y, yp, ypp,  uP, upp, r, rp, rpp, h, rm2, rma, *rpow;
        RSERIES rw, rm1;
        INTERVL i1;
        BND err, unr;



        double t;
        int m;
        int i, j, k;
        unsigned size;
```

hinf

```
t = U.f[U.n-60].center.b;
m = 10;

Le = cvtdi(295.0);
UL = grseval(U,Le);

y = yinf(t, m);
y.center = cvtdb(t);
y.r = b0ero;
ypp = rspower(y, ration(3,2));
yp = rs(y.p.deg, y.center, y.r);
uP = rs(y.p.deg, y.center, y.r);
upp = rs(y.p.deg, y.center, y.r);

for(i=1; i <= ypp.p.deg; ++i){
        yp.p.p[i] = mult(divi(ypp.p.p[i],plus(ifour,
            mult(cvtinti(i),ALPHA))),ifour);
        uP.p.p[i] = divi(plus(y.p.p[i],mult(imthree,yp.p.p[i])),
            imtwo);
        upp.p.p[i] = plus(mult(ypp.p.p[i],itwo),neg(yp.p.p[i]));
}
yp.g = umultb(ypp.g, udivb(bfour, lplusb(bfour,lcvtib(
    mult(itwo,ALPHA)))));
yp.h = umultb(ypp.h, udivb(bfour, lplusb(bfour,lmultb(cvtintb(
    ypp.p.deg+1),lcvtib(ALPHA)))));

uP.g = udivb(uplusb(y.g,umultb(yp.g,bthree)),btwo);
uP.h = udivb(uplusb(y.h,umultb(yp.h,bthree)),btwo);

upp.g = uplusb(yp.g,umultb(ypp.g,btwo));
upp.h = uplusb(yp.h,umultb(ypp.h,btwo));

if(grtb(bl1nrs(uP),bhalf)){
        printf("HINF: Derivative is not bounded below by 1/2\n ");
        abort();
}

if(grtb(bl1nrs(upp),bone)){
        printf("HINF: u is not convex\n");
```

```
        abort();
    }

    r = rs(y.p.deg, y.center, y.r);

    r.p.p[0] = ione;
    r.p.p[1] = divi(y.p.p[1],itwo);
    r.p.deg = 1;
    while (r.p.deg < y.p.deg ){
        rma = rspower(r,neg(ALPHA));
        rpow =  (RSERIES *)calloc(size=r.p.deg+1,sizeof(RSERIES));
        rsmatpower(rma, rpow);
        rw = rs(r.p.deg+1,r.center,r.r);
        for(j=1; j <= r.p.deg+1; ++j){
            err= b0ero;
            for(k=1; k <= j; ++k){
                if(k <= r.p.deg)
                    i1 = rpow[k].p.p[j-k];
                else
                    i1 = ione;
                rw.p.p[j] = plus(mult(i1,y.p.p[k]),rw.p.p[j]);
                if(k > 1) err = maxb(err, uabs(i1));
            }
            rw.p.p[j] = ienlarge(rw.p.p[j],umultb(err,y.g));
        }


        rw.p.p[0] = ione;
        r.p.deg++;
        rm2 = rspower(r, imtwo);
        for(j=0; j <= r.p.deg; ++j){
            r.p.p[r.p.deg] = plus(r.p.p[r.p.deg],
                  mult(rm2.p.p[j], rw.p.p[r.p.deg-j]));
        }
        r.p.p[r.p.deg] = divi(r.p.p[r.p.deg], itwo);

        for(j=0; j <= r.p.deg-1; ++j) freep(rpow[j].p);
        freep(rm2.p), freep(rma.p), free((char *)rpow), rpow = NULL;
        freep(rw.p);
    }
```

```
rma = rspower(r,neg(ALPHA));
rm2 = rspower(r,imone);
rm2 = rsmultf(rm2,rm2);
rpow = (RSERIES *)calloc(size=r.p.deg+1,sizeof(RSERIES));
rsmatpower(rma, rpow);
rw = rs(r.p.deg,r.center,r.r);

rw.p.p[0] = ione;
err = b0ero;
for(j=1; j <= r.p.deg; ++j){
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k)
                rpow[j].h = uplusb(rpow[j].h,uabs(rpow[j].p.p[k]));
        err = maxb(err, rpow[j].h);
        for(k = r.p.deg; k >= j; --k)
                rpow[j].p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
                rpow[j].p.p[k] = izero;
        rw = rsplusf(rw, rsca(rpow[j],y.p.p[j]));
}
rw.h = uplusb(rw.h,umultb(err,y.g));

for(j=2; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=2; k <= j; ++k){
                err = maxb(err,uabs(rpow[k].p.p[j]));
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(y.g,err));
}



r.p.p[0] = izero;
unr = bl1nrs(r);
if (unr.b >= (double) 1){
        printf("HINF: error no. 1\n");
        abort();
}
unr = lplusb(bone,negb(unr));
unr = uabs(poweri(cvtbi(unr),mult(cvtinti(-r.p.deg-1),ALPHA)));
```

```
rw.h = uplusb(rw.h, umultb(y.g, unr));
r.p.p[0] = ione;

rw = rsmult(rw,rm2);
r.h = rw.h;
freep(rw.p);

freep(rm2.p), freep(rma.p);
for(j=0; j <= r.p.deg; ++j)freep(rpow[j].p);

rma = rspower(r,neg(ALPHA));
rm2 = rsmult(r,r);
rm2 = rsmultf(rm2,rscopy(r));
rsmatpower(rma, rpow);
rw = rs(r.p.deg,r.center,r.r);
rpp = rs(r.p.deg,r.center,r.r);

rw.p.p[0] = ione;
err = b0ero;
for(j=1; j <= r.p.deg; ++j){
        rpp.h = rpow[j].h;
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k){
                rpp.h = uplusb(rpp.h,uabs(rpow[j].p.p[k]));
        }
        err = maxb(err, rpp.h);

        for(k = r.p.deg; k >= j; --k)
                rpp.p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
                rpp.p.p[k] = izero;
        rw = rsplusf(rw,
            rsca(rpp,uP.p.p[j]));

}
rw.h = uplusb(rw.h,umultb(err,uP.g));

for(j=2; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=2; k <= j; ++k){
                err = maxb(err,uabs(rpow[k].p.p[j-k]));
```

```
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(uP.g,err));
}

r.p.p[0] = izero;
unr = bl1nrs(r);
if (unr.b >= (double) 1){
        printf("HINF: error no. 3\n");
        fflush(stdout);
        abort();
}

unr = lplusb(bone,negb(unr));

err = labsi(divi(cvtinti(12),poweri(UL,ihalf)));
if(lsb(lmultb(err,unr),cvtdb(t))){
        printf("HINF: wrong choice of L\n");
        printf("err = ");
        printbd(err);
        printf("unr = ");
        printbd(unr);
        printf(" UL = ");
        printival(UL);
        printf("M = %e",t);
        printrs(r);
        abort();
}

unr = uabs(poweri(cvtbi(unr), mult(cvtinti(-r.p.deg-1),ALPHA)));
rw.h = uplusb(rw.h,umultb(unr,uplusb(uP.h, uP.g)));
rw.g = b0ero;
r.p.p[0] = ione;


rp = rsmultf(rspowerf(rw,imone),rm2);
rp.p.p[0] = ione;
freep(rpp.p), freep(rma.p);

rm1 = rspower(r, imone);
rm2 = rsmult(rm1,rm1);
```

```
rm2 = rsmultf(rm2,rm2);

rw = rs(r.p.deg, r.center, r.r);
rw.p.p[0] = ione;
err = b0ero;
for(j=1; j <= r.p.deg; ++j){
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k)
                rpow[j].h = uplusb(rpow[j].h,uabs(rpow[j].p.p[k]));
        err = maxb(err,rpow[j].h);
        for(k = r.p.deg; k >= j; --k)
                rpow[j].p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
                rpow[j].p.p[k] = izero;
        rw = rsplusf(rw,
            rsca(rpow[j],upp.p.p[j]));
}

rw.h = uplusb(rw.h,umultb(err,upp.g));



for(j=2; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=2; k <= j; ++k){
                err = maxb(err,uabs(rpow[k].p.p[j]));
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(upp.g,err));
}


rw.h = uplusb(rw.h,umultb(unr,uplusb(upp.h, upp.g)));
rw.g = b0ero;

rpp = rsmultf(rm2,rw);
rpp = rsmultf(rsmult(rp,rp),rpp);
rpp = rsmultf(rpp, rscopy(rp));
rpp.p.p[0] = ione;


rw = rsmult(rp,rm1);
```

```
        h = rsplusf(rsca(rw,itwo),
             rsplusf(rscaf(rsmult(rpp,rm1),imthree),rsmult(rw,rw)));

        freep(y.p), freep(yp.p), freep(ypp.p);
        freep(uP.p), freep(upp.p);
        freep(r.p), freep(rp.p), freep(rpp.p);
        freep(rm1.p), freep(rw.p);
        for(j=0; j <= r.p.deg; ++j) freep(rpow[j].p);
        free((char *)rpow), rpow = NULL;


        return(h);
}
```

h_at_0

```
RSERIES
h_at_0()
{
        RSERIES y, yp, ypp,  uP, upp, r, rp, rpp, h, rma, *rpow;
        RSERIES rw, rm1;
        INTERVL sc, i1;
        BND err, unr;
        int i, j, k;
        unsigned size;
        double t;
        int m = 20;

        De = cvtdi(0.0099);
        UD = grseval(U,De);

        i = 0;
        t = 0.012;

        y = y_at_0(t, m);
        ypp = rspower(y, ration(3,2));
        yp = rs(y.p.deg+1, y.center, y.r);
        uP = rs(y.p.deg, y.center, y.r);
        yp.p.p[0] = neg(W);
```

```
ypp.p.p[0] = ione;

sc = poweri(cvtdi(t),ihalf);
for(i=1; i <= yp.p.deg; ++i)
        yp.p.p[i] = mult(divi(mult(ypp.p.p[i−1],itwo),cvtinti(i)),sc);
yp.g = umultb(ypp.g, btwo);
yp.g = umultb(yp.g, ucvtib(sc));
yp.g = udivb(yp.g,bthree);
yp.h = udivb(ypp.h, ldivb(cvtintb(yp.p.deg+1),btwo));
yp.h = umultb(yp.h, ucvtib(sc));

for(i=2; i <= y.p.deg; ++i)
        uP.p.p[i] = plus(y.p.p[i],mult(yp.p.p[i−2],cvtdi(t)));
uP.g = uplusb(y.g,umultb(yp.g,cvtdb(t)));
for(i=y.p.deg−1; i <= yp.p.deg; ++i)
        uP.h = uplusb(uabs(yp.p.p[i]),uP.h);
uP.h = uplusb(y.h,umultb(uplusb(yp.h,uP.h),cvtdb(t)));

if(grtb(bl1nrs(uP),bhalf)){
        printf("H_AT_0: Derivative too small !!!\n");
        abort();
}
uP.p.p[0] = ione;

upp = rsca(yp,itwo);
for(i=1; i <= upp.p.deg; ++i)
        upp.p.p[i] = plus(upp.p.p[i], mult(ypp.p.p[i−1],sc));
upp.g = uplusb(upp.g,umultb(ypp.g,ucvtib(sc)));
upp.h = uplusb(upp.h,umultb(ypp.h,ucvtib(sc)));


r = rs(y.p.deg, y.center, y.r);

r.p.p[0] = ione;
r.p.p[2] = neg(y.p.p[2]);
r.p.deg = 2;
while (r.p.deg < y.p.deg ){
        rma = rspower(r,ihalf);
        rpow = (RSERIES *)calloc(size=r.p.deg+1,sizeof(RSERIES));
        for(j=0; j <= r.p.deg; ++j) rpow[j] = rs(r.p.deg,r.center,r.r);
```

```
        rsmatpower(rma, rpow);
        rw = rs(r.p.deg+1,r.center,r.r);
        for(j=1; j <= r.p.deg+1; ++j){
                err= b0ero;
                for(k=1; k <= j; ++k){

                        if(k <= r.p.deg)
                                i1 = rpow[k].p.p[j−k];
                        else
                                i1 = ione;

                        rw.p.p[j] = plus(mult(i1,y.p.p[k]),rw.p.p[j]);
                        if(k > 1) err = maxb(err, uabs(i1));
                }
                rw.p.p[j] = ienlarge(rw.p.p[j],umultb(err,y.g));
        }


        rw.p.p[0] = ione;
        r.p.deg++;
        for(j=0; j < r.p.deg; ++j){
                r.p.p[r.p.deg] = plus(r.p.p[r.p.deg],
                    mult(r.p.p[j], rw.p.p[r.p.deg−j]));
        }
        r.p.p[r.p.deg] = neg(r.p.p[r.p.deg]);

        for(j=0; j <= r.p.deg−1; ++j) freep(rpow[j].p);
        freep(rma.p), free((char *)rpow), rpow = NULL;
        freep(rw.p);
}

rma = rspower(r,ihalf);
rpow = (RSERIES *)calloc(size=r.p.deg+1,sizeof(RSERIES));
for(j=0; j <= r.p.deg; ++j) rpow[j] = rs(r.p.deg,r.center,r.r);
rsmatpower(rma, rpow);
rw = rs(r.p.deg,r.center,r.r);

rw.p.p[0] = ione;
err = b0ero;
for(j=1; j <= r.p.deg; ++j){
```

```
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k)
            rpow[j].h = uplusb(rpow[j].h,uabs(rpow[j].p.p[k]));
        err = maxb(err,rpow[j].h);

        for(k = r.p.deg; k >= j; --k)
            rpow[j].p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
            rpow[j].p.p[k] = izero;
        rw = rsplusf(rw,rsca(rpow[j],y.p.p[j]));
    }
    rw.h = uplusb(rw.h,umultb(err,y.g));
    for(j=2; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=2; k <= j; ++k){
            err = maxb(err,uabs(rpow[k].p.p[j]));
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(y.g,err));
    }

    unr = bl1nrs(r);
    unr = uabs(poweri(cvtbi(unr),divi(cvtinti(r.p.deg+1),itwo)));
    rw.h = uplusb(rw.h, umultb(y.g, unr));

    rw = rsmultf(rw,rscopy(r));

    r.h = umultb(rw.h,btwo);
    freep(rw.p);

    freep(rma.p), free((char *)rpow), rpow = NULL;

    rma = rspower(r,ihalf);
    rpow =  (RSERIES *)calloc(size=r.p.deg+1,sizeof(RSERIES));
    for(j=0; j <= r.p.deg; ++j) rpow[j] = rs(r.p.deg,r.center,r.r);
    rsmatpower(rma, rpow);
    rw = rs(r.p.deg,r.center,r.r);
    rpp = rs(r.p.deg,r.center,r.r);

    rw.p.p[0] = ione;
    err = b0ero;
    for(j=1; j <= r.p.deg; ++j){
```

```
        rpp.h = rpow[j].h;
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k)
                rpp.h = uplusb(rpp.h,uabs(rpow[j].p.p[k]));
        err = maxb(err,rpp.h);
        for(k = r.p.deg; k >= j; --k)
                rpp.p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
                rpp.p.p[k] = izero;
        rw = rsplusf(rw,
            rsca(rpp,uP.p.p[j]));
}
rw.h = uplusb(umultb(err, uP.g), rw.h);


for(j=2; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=2; k <= j; ++k){
                err = maxb(err,uabs(rpow[k].p.p[j-k]));
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(uP.g,err));
}


unr = bl1nrs(r);
err = umultb(ucvtib(UD),unr);
if(grtb(err,cvtdb(t))){
        printf("H_at_0: wrong choice of D\n");
        abort();
}
unr = uabs(poweri(cvtbi(unr), divi(cvtinti(r.p.deg+1),itwo)));
rw.h = uplusb(rw.h,umultb(unr,uplusb(uP.h, uP.g)));
rw.g = b0ero;


rp = rspowerf(rw,imone);
rp.p.p[0] = ione;
freep(rpp.p), freep(rma.p);

rm1 = rspower(r, imone);
```

```
rw = rs(r.p.deg, r.center, r.r);
rw.p.p[0] = upp.p.p[0];
err = b0ero;
for(j=1; j <= r.p.deg; ++j){
        for(k=r.p.deg-j+1; k <= r.p.deg; ++k)
                rpow[j].h = uplusb(rpow[j].h,uabs(rpow[j].p.p[k]));
        err = maxb(err,rpow[j].h);
        for(k = r.p.deg; k >= j; --k)
                rpow[j].p.p[k] = rpow[j].p.p[k-j];
        for(k = 0; k < j; ++k)
                rpow[j].p.p[k] = izero;
        rw = rsplusf(rw,rsca(rpow[j],upp.p.p[j]));
}
rw.h = uplusb(rw.h,umultb(err,upp.g));


for(j=3; j <= rw.p.deg; ++j){
        err = b0ero;
        for(k=3; k <= j; ++k){
                err = maxb(err,uabs(rpow[k].p.p[j]));
        }
        rw.p.p[j] = ienlarge(rw.p.p[j],umultb(upp.g,err));
}

rw.h = uplusb(rw.h,umultb(unr,uplusb(upp.h,upp.g)));
rw.g = b0ero;

rpp = rsmultf(rsmult(rp,rp),rw);
rpp = rsmultf(rpp, rsca(rp,imone));
rpp.p.p[0] = mult(itwo,W);



rw = rsmult(rp,rm1);
h = rsminusf(rsmult(rw,rw), rw);

h.p.deg = h.p.deg - 2;
for(i=0; i <= h.p.deg; ++i)h.p.p[i] = divi(h.p.p[i+2],cvtdi(t));
h.h = udivb(h.h,cvtdb(t));
```

```
	h = rsminusf(rsmult(rpp,rm1), h);

	freep(y.p), freep(yp.p), freep(ypp.p);
	freep(uP.p), freep(upp.p);
	freep(r.p), freep(rp.p), freep(rpp.p);
	freep(rm1.p);
	for(j=0; j <= r.p.deg; ++j) freep(rpow[j].p);
	free((char *)rpow), rpow = NULL;

	return(rstruncf(h,9));
}
```

printh

```
void
printh()
{
	RSERIES h;
	printrsio(hinf());
	h = h_at_0();
	printrsio(h);
	printivalio(Le);
	printivalio(UL);
	printivalio(De);
	printivalio(UD);
}
```

readh

```
void
readh()
{
	HINF = readrsio();
	H0 = readrsio();
	Le = readivalio();
	UL= readivalio();
	De = readivalio();
	UD = readivalio();
}
```

tfint1

```
INTERVL
tfint1(alpha,x)
```

```
INTERVL alpha, x;
{
        INTERVL sol;
        int m, i;
        BND err;

        m = 100;
        if( x.up >= one || x.dn < zero ){
                printf("TFINT1: error\n");
                fflush(stdout);
                abort();
        }

        sol = izero;
        err = udivb(btwo,cvtintb(2*m+3));
        for(i=m; i >= 1; i--){
                sol = mult(plus(sol,inv(plus(cvtinti(i),ihalf))),
                    divi(mult(x,plus(alpha,cvtinti(-i+1))),cvtinti(i)));
                err = udivb(
                    umultb(umultb(uabs(x),err),uplusb(uabs(alpha),
                    cvtintb(i-1))),
                    cvtintb(i));
        }

        sol = mult(plus(sol, itwo), poweri(x,ihalf));
        err = umultb(err,udivb(umultb(uplusb(cvtintb(m),negb(lcvtib(alpha))),
            uabs(ratpower(x,3,2))),cvtintb(m+1)));
        return(ienlarge(sol, err));
}
```

tfint2

```
INTERVL
tfint2(alpha, a, b)
INTERVL a, b, alpha;
{
        INTERVL sol;

        sol = poweri(iunion(a,b),alpha);
```

```
        sol = mult(sol, mult(itwo,plus(b,neg(a))));
        sol = divi(sol,
            plus(poweri(plus(b,imone),ihalf),poweri(plus(a,imone),ihalf)));

        return(sol);
}
```

tfint3

```
INTERVL
tfint3(alpha, a, b)
INTERVL a, b, alpha;
{
        INTERVL sol;

        sol = mult(poweri(plus(iunion(a,b),imone),ihalf),plus(alpha,ione));
        sol = divi(plus(poweri(b,plus(alpha,ione)),
            neg(poweri(a,plus(alpha,ione)))),sol);
        return(sol);
}
```

tfint4

```
INTERVL
tfint4(alpha, a, b)
INTERVL a, alpha, b;
{
        RSERIES rw1, rw2;
        BND x, r;

        x.b = (a.dn+b.up)/2.0;
        r = maxb(uplusb(x,negb(lcvtib(a))),uplusb(negb(x),ucvtib(b)));
        rw1 = rslinpower(ione,imone,neg(ihalf),8,x,r);
        rw2 = rslinpower(ione,izero,alpha,8,x,r);
        return(rsdintf(rsmultf(rw1,rw2),b,a));
}
```

tfintf1

```
INTERVL
tfintf1(alpha,x)
INTERVL alpha, x;
{
        INTERVL sol, a, b;
        double step;
```

```
	if(x.up < 1.8)return(tfint1(alpha,plus(x,imone)));

	b.up = b.dn = 1.8;
	sol = tfint1(alpha, plus(b,imone));

	step = x.up/100.0;
	while(b.up < x.dn){
		a = b;
		b.up = b.dn = b.dn + step;
		a = tfint2(alpha,a,b);
		sol = plus(sol,a);
	}

	return(plus(sol,tfint2(alpha,b,x)));
}
```

tfintf2

```
INTERVL
tfintf2(alpha,x)
INTERVL alpha, x;
{
	INTERVL sol, a, b;
	double step;

	if(x.up < 1.8)return(tfint1(alpha,plus(x,imone)));

	b.up = b.dn = 1.8;
	sol = tfint1(alpha, plus(b,imone));

	step = x.up/100.0;
	while(b.up < x.dn){
		a = b;
		b.up = b.dn = b.dn + step;
		a = tfint4(alpha,a,b);
		sol = plus(sol,a);
	}

	return(plus(sol,tfint2(alpha,b,x)));
}
```

secder0_sp

```
INTERVL
secder0_sp(w)
INTERVL w;
{
        INTERVL a, sol, uLwm2, m, upL, ta;
        int i;

        m = cvtbi(H0.r);
        upL = grsdereval(U,De);
        uLwm2 = divi(UD,square(w));
        if(uLwm2.dn < one){
                printf("SECDER0_sp: Omega is too large\n");
                abort();
        }
        ta = poweri(m,neg(ihalf));

        sol = izero;

        a = divi(cvtinti(H0.p.deg+1),itwo);
        sol = izero;
        sol = ienlarge(sol,umultb(uabs(mult(w,ta)),
            umultb(H0.h,uabs(tfintf1(a,uLwm2)))));
        for(i=H0.p.deg; i >= 0; --i){
                a = divi(cvtinti(i),itwo);
                if(i > 1) a = tfintf1(a,uLwm2);
                else a = tfintf2(a,uLwm2);

                sol = mult(plus(mult(H0.p.p[i],a),mult(sol,ta)),w);
        }

        sol = plus(divi(mult(imtwo,UD),mult(mult(De,upL),poweri(
            plus(UD,neg(square(w))),ihalf))), mult(sol,itwo));
        return(sol);
}


INTERVL
```

secder1_sp

```
secder1_sp(w)
INTERVL w;
{
        INTERVL wa, a, sol, a2, uLwm2, m, upL, ta;
        int i;

        m = cvtbi(HINF.center);
        upL = grsdereval(U,Le);
        uLwm2 = divi(UL,square(w));
        if(uLwm2.dn < one){
                printf("SECDER1_sp: Omega is too large\n");
                abort();
        }
        a2 = divi(ALPHA,itwo);
        wa = poweri(w,ALPHA);
        ta = mult(wa, poweri(divi(m,cvtinti(12)),ALPHA));


        a = plus(mult(cvtinti(HINF.p.deg+1),a2),imone);
        sol = izero;
        sol = ienlarge(sol,
            umultb(uabs(ta),umultb(HINF.h,uabs(tfintf1(a,uLwm2)))));
        for(i=HINF.p.deg; i >= 1; --i){
                a = plus(mult(cvtinti(i),a2),imone);
                if( i > 1) a = tfintf1(a,uLwm2);
                else a = tfintf2(a,uLwm2);
                sol = mult(plus(sol,mult(HINF.p.p[i],a)),ta);
        }
        sol = divi(divi(sol,itwo),w);
        sol = plus(divi(mult(itwo,UL),mult(mult(Le,upL),poweri(
            plus(UL,neg(square(w))),ihalf))), sol);
        return(sol);
}
```

secder0_speps

```
INTERVL
secder0_speps(w)
```

```
INTERVL w;
{
        INTERVL r, sol;
        int i;


        r = poweri(divi(UD,cvtbi(H0.r)),ihalf);
        sol = cvtbi(H0.h);
        for(i=H0.p.deg; i >= 0; --i){
                sol = mult(sol,r);
                if(H0.p.p[i].dn < (double) 0)
                        sol = plus(sol,iabs(H0.p.p[i]));
        }
        sol = mult(sol,poweri(UD,ihalf));
        sol = mult(sol,cvtinti(4));
        sol = plus(divi(mult(itwo,UD),mult(mult(De,grsdereval(U,De)),poweri(
            plus(UD,neg(square(w))),ihalf))),sol);
        return(sol);
}
```

secder1_speps

```
INTERVL
secder1_speps(w)
INTERVL w;
{
        INTERVL r, sol, i1;
        int i;

        r = mult(poweri(UL,ihalf),divi(cvtbi(HINF.center),cvtinti(12)));
        r = poweri(r,ALPHA);

        sol = mult(cvtbi(HINF.h),r);
        if(HINF.p.p[2].dn < (double) 0){
                printf("SECDER1_SPEPS: second term negative.\n");
                abort();
        }
        for(i=HINF.p.deg; i >= 2; --i){
                if(HINF.p.p[i].dn < (double) 0)
                        sol = plus(sol,iabs(HINF.p.p[i]));
                sol = mult(sol,r);
```

```
	}
	sol = mult(sol,r);
	sol = divi(sol, poweri(UL,ihalf));



	i1 = mult(Le,iabs(grsdereval(U,Le)));
	i1 = mult(i1,poweri(plus(UL,neg(square(w))),ihalf));
	sol = plus(divi(mult(itwo,UL), i1), sol);
	return(sol);
}
```

secder_eps

```
INTERVL
secder_eps()
{
	GRS yw;

	yw = grstimesx(grstimesx(grstimesx(YRS)));
	yw = grspowerf(yw,neg(ihalf));
	return(grsdintf(yw,De,Le));
}
```

```
#include <math.h>
#include <stdio.h>

BND
lipreg(u0,u1,x0,r,a)
INTERVL u0, u1, x0;
BND r, a;
{
        BND g1, sol, g0, c1, c2;

        g0 = umultb(r,uabs(u1));
        g1 = udivb(g0,lcvtib(u0));
        g0 = udivb(uplusb(g0,a),lcvtib(u0));
        c1 = umultb(frak22(neg(ihalf),udivb(r,lcvtib(x0))),ucvtib(ratpower(x0,
            -1,2)));
        c2 = umultb(ucvtib(ratpower(u0,1,2)),
            frac22(ration(3,2),g0));

        c1 = udivb(umultb(usquareb(r),c1),btwo);
        sol = umultb(c1,c2);
        c2 = umultb(ucvtib(ratpower(u0,3,2)),
            frak22(ration(3,2),g1));

        c1 = umultb(c1, c2);
        c2 = lmultb(a,lplusb(bone,negb(sol)));
        if (lseqb(c1,c2)) return(sol);
        else{
                return(bone);
        }
}
```

lipreg

```
BND
lip0(w,r,a)
INTERVL w;
BND r, a;
{
        BND g1, sol, g0, c1, c2;
```

lip0

```
	g1 = umultb(r,uabs(w));
	g0 = uplusb(g1,a);
	sol = frac22(ration(3,2),g0);
	c2 = frak22(ration(3,2),g1);
	c1 = udivb(umultb(bfour,ucvtib(ratpower(cvtbi(r),3,2))),cvtintb(3));

	sol = umultb(sol, c1);

	c1 = umultb(c1,c2);
	c2 = lmultb(a,lplusb(bone,negb(sol)));

	if (lseqb(c1,c2)) return(sol);
	else{
		return(bone);
	}
}
```

vtffx

```
double
vtffx(xin, xout, u0, u1,y0, y1)
double xin, u0, u1;
INTERVL xout, *y0, *y1;
{
	RSERIES y, yp, ypp, rsw;
	INTERVL ithreehalfs, r;
	BND bn, eps;
	int i;
	double p[SIZE], pw[SIZE];

	pzer(p), pzer(pw);
	p[0] = u0, p[1] = u1*fabs(.5*(xout.dn+xout.up)−xin);

	pw[0] = xin;
	pw[1] = fabs(.5*(xout.up+xout.dn)−xin);
	myprpower(pw, −0.5, pw);
```

```
for(i=1; i <= DEGREE; ++i){
        myprpower(p,1.5,p);
        pprod(p,pw,p);
        pinte(p,p);
        pinte(p,p);
        psca(p,(.5*(xout.up+xout.dn)-xin)*(.5*(xout.up+xout.dn)-xin),p);
        p[0] = u0, p[1] = u1*fabs(.5*(xout.up+xout.dn)-xin);
}

r = iabs(plus(xout,cvtdi(-xin)));

y = rs(DEGREE,cvtdb(xin),cvtdb(r.up));
for(i=2; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);

eps = bl1nrs(y);
bn = lipreg(cvtdi(u0), cvtdi(u1), cvtdi(xin), ucvtib(r), eps);
while(grteqb(bn,bone)){
        eps = maxb(eps, cvtdb(eps.b * 1.1));
        bn = lipreg(cvtdi(u0), cvtdi(u1), cvtdi(xin), ucvtib(r), eps);
}

y.p.p[0] = cvtdi(u0);
y.p.p[1] = mult(cvtdi(r.up),cvtdi(u1));

rsw = rs(DEGREE,cvtdb(xin),cvtdb(r.up));
rsw.p.p[0] = cvtdi(xin);
rsw.p.p[1] = cvtdi(r.up);
rsw = rspowerf(rsw,neg(ihalf));

ithreehalfs = divi(ithree,itwo);
yp = rspower(y,ithreehalfs);
ypp = rsintegf(rsintegf(rsmultf(yp,rscopy(rsw))));

yp = rscopy(y);
yp.p.p[0] = izero;
yp.p.p[1] = izero;
eps = bl1nrsf(rsminusf(yp,ypp));

y.g = udivb(eps,lplusb(bone, negb(bn)));
y.k = 2;
```

```
	rsw.k = 2;
	*y0 = rseval(y,xout);

	*y1 = rsevalf(rsintegf(rsmultf(rspower(y,ithreehalfs),rsw)), xout);



	*y1 = plus(*y1,cvtdi(u1));
	freep(y.p);
	return(eps.b);
}


double
supervtffx(xin, xout, u0, u1,y0, y1)
```

supervtffx

```
double xin;
INTERVL xout, u0, u1;
INTERVL *y0, *y1;
{
	INTERVL yw0, yw1;
	double eps;

	if(xin <= xout.dn){
		eps = vtffx(xin, xout, u0.up, u1.up, &yw0, &yw1);
		y0->up = yw0.up, y1->up = yw1.up;
		eps += vtffx(xin, xout, u0.dn, u1.dn, &yw0, &yw1);
		y0->dn = yw0.dn, y1->dn = yw1.dn;
		return(eps);
	}
	else if(xin >= xout.up){
		eps = vtffx(xin, xout, u0.dn, u1.up, &yw0, &yw1);
		y0->dn = yw0.dn, y1->up = yw1.up;
		eps += vtffx(xin, xout, u0.up, u1.dn, &yw0, &yw1);
		y0->up = yw0.up, y1->dn = yw1.dn;
		return(eps);
	}
	else{
		printf("SUPERVTFFX: case not considered\n\n\n");
		fflush(stdout);
		abort();
```

```
        }
}

RSERIES
vtffxi(xin, xout, u0, u1)
INTERVL xin, xout, u0, u1;
{
        RSERIES y, yp, ypp, rsw;
        INTERVL ithreehalfs, r;
        BND bn, eps;
        int i;
        double p[SIZE], pw[SIZE];

        pzer(p), pzer(pw);
        p[0] = .5*(u0.dn+u0.up);
        p[1] = .5*(u1.up+u1.dn)*fabs(.5*(xout.up+xout.dn)
            -0.5*(xin.up+xin.dn));

        pw[0] = 0.5*(xin.up+xin.dn);
        pw[1] = fabs(.5*(xout.up+xout.dn)-0.5*(xin.up+xin.dn));
        myprpower(pw, -0.5, pw);

        for(i=1; i <= DEGREE; ++i){
                myprpower(p,1.5,p);
                pprod(p,pw,p);
                pinte(p,p);
                pinte(p,p);
                psca(p,(.5*(xout.up+xout.dn)-0.5*(xin.up+xin.dn))
                    *(.5*(xout.up+xout.dn)-0.5*(xin.up+xin.dn)),p);
                p[0] = .5*(u0.dn+u0.up);
                p[1] = .5*(u1.up+u1.dn)*fabs(.5*(xout.up+xout.dn)
                    -0.5*(xin.up+xin.dn));
        }

        r = iabs(plus(xout,neg(xin)));

        y = rs(DEGREE,b0ero,cvtdb(r.up));
        y.p.p[0] = u0;
        y.p.p[1] = mult(cvtdi(r.up),u1);
```

vtffxi

```
        for(i=2; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);

        rsw = rs(DEGREE,b0ero,cvtdb(r.up));
        rsw.p.p[0] = xin;
        rsw.p.p[1] = cvtdi(r.up);
        rsw = rspowerf(rsw,neg(ihalf));

        ithreehalfs = divi(ithree,itwo);
        yp = rspower(y,ithreehalfs);
        ypp = rsintegf(rsintegf(rsmultf(yp,rsw)));

        yp = rscopy(y);
        yp.p.p[0] = izero;
        yp.p.p[1] = izero;

        eps = bl1nrs(yp);
        bn = lipreg(u0, u1, xin, ucvtib(r), eps);
        while(grteqb(bn,bone)){
                eps = maxb(eps,cvtdb(eps.b * 1.1));
                bn = lipreg(u0, u1, xin, ucvtib(r), eps);
        }

        eps = bl1nrsf(rsminusf(yp,ypp));
        y.g = udivb(eps,lplusb(bone, negb(bn)));
        y.k = 2;
        return(y);
}
POLY
tfypoly(u0, u1, r, i)
INTERVL u0, u1, r;
int i;
{
        POLY poly, res;
        POLY rsw;
        int j, k;
        poly = make_poly(i);
        poly.p[0] = u0, poly.p[1] = u1;

        rsw = make_poly(i);
```

tfypoly

```
        rsw.p[0] = r;
        rsw.p[1] = ione;
        rsw = polypowerf(rsw,neg(ihalf));

        for(k=2; k <= i; ++k){
                res = make_poly(k-1);
                for(j=0; j<=k-2; ++j) res.p[j] = poly.p[j];
                res = polypowerf(res, ration(3,2));
                res.p[k-2] = coeffmult(res,rsw,k-2);
                poly.p[k] = divi(res.p[k-2], cvtinti(k*(k-1)));
                freep(res);
        }

        freep(rsw);
        return(poly);
}
```

vtffxi2

```
RSERIES
vtffxi2(xin, xout, u0, u1)
INTERVL xin, xout, u0, u1;
{
        RSERIES y, yp, ypp, rsw;
        INTERVL ithreehalfs, r;
        BND bn, eps;
        int i;
        POLY poly;


        r = iabs(plus(xout,neg(xin)));
        poly = polyscalef(tfypoly(u0, u1, xin, DEGREE), cvtdi(r.up));

        y = rs(DEGREE,b0ero,cvtdb(r.up));
        y.p.p[0] = u0;
        y.p.p[1] = mult(cvtdi(r.up),u1);
        for(i=2; i<= DEGREE; ++i)
                y.p.p[i] = cvtdi(.5*(poly.p[i].up+poly.p[i].dn));

        rsw = rs(DEGREE,b0ero,cvtdb(r.up));
        rsw.p.p[0] = xin;
```

```
	rsw.p.p[1] = cvtdi(r.up);
	rsw = rspowerf(rsw,imhalf);

	ithreehalfs = divi(ithree,itwo);
	yp = rspower(y,ithreehalfs);
	ypp = rsintegf(rsintegf(rsmultf(yp,rsw)));

	yp = rscopy(y);
	yp.p.p[0] = izero;
	yp.p.p[1] = izero;

	eps = bl1nrs(yp);
	bn = lipreg(u0, u1, xin, ucvtib(r), eps);
	while(grteqb(bn,bone)){
		eps = maxb(eps,cvtdb(eps.b * 1.1));
		bn = lipreg(u0, u1, xin, ucvtib(r), eps);
	}

	eps = bl1nrsf(rsminusf(yp,ypp));
	y.h = udivb(eps,lplusb(bone, negb(bn)));
	for(i=2; i<= DEGREE; ++i)
		y.p.p[i] = iunion(y.p.p[i], poly.p[i]);

	return(y);
}
```

vtff0

```
void
vtff0(w,t,y0, y1)
double w;
BND t;
INTERVL *y0, *y1;
{
	RSERIES y, yp, ypp;
	INTERVL sc;
	INTERVL ithreehalfs;
	BND eps, b0;
	double p[SIZE];
	int i, j;
```

```
pzer(p);
p[0] = 1.0, p[2] = −w;
for(i=0; i <= DEGREE; ++i){
        myprpower(p,1.5,p);
        for(j=DEGREE; j >= 3; −−j)
                p[j] = 4.0*p[j−3]/(j*(j−2));
        p[0] = 1.0, p[1]=zero, p[2]= −w;
}
pscale(p,sqrt(t.b),p);
y = rs(DEGREE,b0ero,t);
for(i=3; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);
b0 = bl1nrs(y);
y.p.p[0] = ione;
y.p.p[2] = mult(cvtdi(−w),cvtbi(t));

ithreehalfs = divi(ithree,itwo);
yp = rspower(y,ithreehalfs);
ypp = rs(DEGREE+3, b0ero, t);
for (i=0; i <= DEGREE; ++i) ypp.p.p[i+3]= divi(yp.p.p[i],
    divi(cvtinti((i+1)*(i+3)),cvtinti(4)));
ypp.h = umultb(yp.h,
    udivb(cvtintb(4),cvtintb((DEGREE+2)*(DEGREE+4))));
sc = iexp(mult(divi(cvtinti(3),cvtinti(2)),ilog(cvtbi(t))));
ypp = rscaf(ypp, sc);
ypp.p.p[0] = y.p.p[0];
ypp.p.p[1] = y.p.p[1];
ypp.p.p[2] = y.p.p[2];

eps = btwo;
while(grteqb(eps,bone)){
        eps = lip0(cvtdi(w),t,b0);
        b0 = maxb(b0,cvtdb(b0.b * 1.01));
}
b0 = eps;
eps = bl1nrsf(rsminus(y,ypp));
y.g = umultb(eps, lplusb(bone,negb(b0)));
*y0 = izero, *y1 = izero;
for(i=0; i<=y.p.deg; ++i){
        *y0 = plus(*y0,y.p.p[i]);
        *y1 = plus(*y1,divi(yp.p.p[i],divi(cvtinti(i+1),itwo)));
```

```
        }
        *y0 = ienlarge(*y0,uplusb(y.g,y.h));
        ypp = rspower(y,ithreehalfs);
        sc = iexp(mult(ihalf,ilog(cvtbi(t))));

        ypp.g = udivb(ypp.g,btwo);
        ypp.h = udivb(ypp.h,cvtintb(2*(ypp.p.deg+1)));
        *y1 = mult(ienlarge(*y1,uplusb(ypp.g,ypp.h)), sc);
        *y1 = plus(*y1,neg(cvtdi(w)));
        freep(y.p), freep(yp.p), freep(ypp.p);
}
```

y_at_0

```
RSERIES
y_at_0(tb,m)
double tb;
int m;
{
        RSERIES y, yp, ypp;
        INTERVL sc;
        INTERVL ithreehalfs;
        BND t, eps, b0;
        double w, p[SIZE];
        int olddeg, i, j;

        t = cvtdb(tb);
        sc = iexp(mult(divi(ithree,itwo ),ilog(cvtbi(t))));
        olddeg = DEGREE;
        DEGREE = m;
        if (m >= SIZE){
                printf("Y_AT_0: DEGREE %d is not possible\n",m);
                abort();
                fflush(stdout);
        }

        pzer(p);
        w = .5*(W.up+W.dn);
        p[0] = 1.0, p[2] = -w*tb;
        for(i=0; i <= DEGREE; ++i){
```

```
            myprpower(p,1.5,p);
            for(j=DEGREE; j >= 3; --j)
                    p[j] = 2.0*(sc.up+sc.dn)*p[j-3]/(j*(j-2));
            p[0] = 1.0, p[1]=zero, p[2]= -w*tb;
    }
    y = rs(DEGREE,b0ero,t);
    for(i=3; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);
    b0 = bl1nrs(y);
    y.p.p[0] = ione;
    y.p.p[2] = neg(mult(W,cvtbi(t)));

    ithreehalfs = divi(ithree,itwo);
    yp = rspower(y,ithreehalfs);
    ypp = rs(DEGREE+3, b0ero, t);
    for (i=0; i <= DEGREE; ++i) ypp.p.p[i+3]= divi(yp.p.p[i],
        divi(cvtinti((i+1)*(i+3)),ifour));
    ypp.h = umultb(yp.h,udivb(cvtintb(4),cvtintb((DEGREE+2)*(DEGREE+4))));
    ypp = rscaf(ypp, sc);

    yp = rscopy(y);
    ypp.p.p[0] = y.p.p[0] = izero;
    ypp.p.p[1] = y.p.p[1] = izero;
    ypp.p.p[2] = y.p.p[2] = izero;

    eps = bl1nrsf(rsminusf(y,ypp));
    while(grteqb(lip0(W,t,b0),bone)) b0 = maxb(b0, cvtdb(b0.b*1.1));
    yp.g = udivb(eps, dengeob(lip0(W,t,b0)));
    DEGREE = olddeg;
    return(yp);
}
```

tfw

```
INTERVL
tfw(w, tol)
INTERVL w;
double tol;
{
    BND b;
```

```
INTERVL i1, i2, i3, i4, r;
double eps, wtest, x;

b.b = 0.008;
while(w.up-w.dn>tol){
        printf("\n\nBOUNDS FOR w:\n");
        printival(w);
        printivalio(w);
        fflush(stdout);

        wtest = .5*(w.up+w.dn);
        vtff0(wtest, b, &i1, &i2);
        r.up = b.b;
        x = 0.0008;
        while(i2.dn<= 0 && r.up < 250.0){
                r.dn = r.up, r.up = r.dn+x;
                if(r.dn < 2.104025275
                    && r.up > 2.104025275)
                        r.up = 2.104025275;
                eps = supervtffx(r.dn, cvtdi(r.up),
                    i1, i2, &i1, &i2);

                i3 = divi(cvtinti(5),cvtinti(2));
                i3 = iexp(mult(i3,ilog(i1)));
                i3 = divi(i3,iexp(mult(ihalf,ilog(cvtdi(r.up)))));
                i4 = square(i2);
                i3 = mult(itwo,i3);
                if(i3.up <= i4.dn) i2.dn = 1.0;

                if(eps   > 5.e-17) x *= 0.7;
                if(eps   > 5.e-16) x *= 0.5;
                if(eps   < 1.e-17) x *= 1.2;
                x = minm(x,0.3);
                if(r.up > 10.0) x = minm(x,0.5);

        }
        if(r.up >= 250.0){
                return(w);
        }
        else if(i3.up <= i4.dn) w.up = wtest;
```

```
            else w.dn = wtest;
        }
        return(w);
}
```

tff

```
GRS
tff(w)
INTERVL w;
{
        GRS y;
        BND b;
        unsigned size;
        INTERVL i1[1000], i2[1000], i3, i4 , i5[1000], i6[1000];
        double eps, x, r[1000];
        int count = 0;
        int count2 = 0;

        b.b = 0.008;
        vtff0(w.up, b, &i1[0], &i2[0]);
        vtff0(w.dn, b, &i5[0], &i6[0]);
        r[count] = b.b;
        x = 0.0008;
        i3.up = 1.0, i4.dn = zero;
        while(i3.up > i4.dn || i6[count].dn < 0){
                if(i3.up <= i4.dn || i6[count].dn >= 0) ++count2;
                r[count+1] =  r[count]+x;
                if(r[count] < 2.10402528 && r[count+1] > 2.10402528)
                        r[count+1]=maxm(r[count],2.10402528);
                if(i3.up > i4.dn )
                        eps = supervtffx(r[count], cvtdi(r[count+1]),
                             i1[count], i2[count], &i1[count+1], &i2[count+1]);
                if(i6[count].dn < 0)
                        eps += supervtffx(r[count], cvtdi(r[count+1]),
                             i5[count], i6[count], &i5[count+1], &i6[count+1]);

                ++count;
                if(i3.up > i4.dn ){
```

```
                        i3 = divi(cvtinti(5),cvtinti(2));
                        i3 = iexp(mult(i3,ilog(i1[count])));
                        i3 = divi(i3,iexp(mult(ihalf,ilog(cvtdi(r[count])))));
                        i4 = square(i2[count]);
                        i3 = mult(itwo,i3);
                }

                if(eps    > 1.e-16) x *= 0.7;
                if(eps    > 1.e-15) x *= 0.5;
                if(eps    < 2.e-17) x *= 1.2;
                x = minm(x,4.0);
                if(r[count] > 200.0) x = minm(x,1.0);
                if(r[count] < 10.0) x = minm(x,0.05);
                if(r[count] > 250.0) x = minm(x,0.3);
        }

        count -= count2;
        y.n = count+1;
        y.f = (RSERIES *)calloc(size=count+2,sizeof(RSERIES));
        y.f[0] = rs(1,b0ero,bone);
        y.f[0].p.p[0] = ione;
        y.f[0].p.p[1] = w;
        for(count=1; count <= y.n; ++count){
                y.f[count] = rs(1,cvtdb(r[count-1]),bone);
                y.f[count].p.p[0].up = i5[count-1].up;
                y.f[count].p.p[0].dn = i1[count-1].dn;
                y.f[count].p.p[1].up = i6[count-1].up;
                y.f[count].p.p[1].dn = i2[count-1].dn;
        }
        return(y);
}
```

tfrs

```
GRS
tfrs(y)
GRS y;
{
        int i;
        GRS sol;
```

```
	unsigned size;
	/*char *calloc();*/
	INTERVL cvtdi(), x1, x2;

	sol.f = (RSERIES *)calloc(size=y.n−1,sizeof(RSERIES));
	sol.n = y.n−2;
	for (i=0; i <= sol.n; ++i){
		x1 = y.f[i+1].p.p[0];
		x2 = y.f[i+1].p.p[1];
		sol.f[i] = vtffxi(cvtdi(y.f[i+1].center.b),
		    cvtdi(y.f[i+2].center.b),x1,x2);
		sol.f[i].center = y.f[i+1].center;
	}

	return(sol);
}
```

Omega

```
INTERVL
Omega(u)
GRS u;
{
	int d;
	INTERVL ievders(), sol, derw;
	double otest, otest1, otest2;

	sol.dn = (double) 2;
	sol.up = (double) 3;

	for(;;){
		otest = .5*(sol.up+sol.dn);
		if(otest == sol.up || otest == sol.dn)
			return(sol);
		derw = grsdereval(u,cvtdi(otest));
		if(derw.up <= zero) sol.up = otest;
		else if(derw.dn >= zero) sol.dn = otest;
		else
		{
			otest1 = otest;
			d = 1;
```

```
                    while(d ){
                              otest2 = .5*(otest+sol.up);
                              if(otest2 == otest || otest2 == sol.up)d = 0;
                              derw = grsdereval(u,cvtdi(otest2));
                              if(derw.up <= zero ) sol.up = otest2;
                              else otest = otest2;
                    }

                    otest = otest1;
                    d = 1;
                    while(d ){
                              otest2 = .5*(otest+sol.dn);
                              if(otest2 == otest || otest2 == sol.dn)d = 0;
                              derw = grsdereval(u,cvtdi(otest2));
                              if(derw.up >= zero ) sol.dn = otest2;
                              else otest = otest2;
                    }

                    return(sol);
          }
     }
}

void
tfprint()
{
     GRS grstimesx();
     INTERVL  grseval();

     printivalio(W);
     printivalio(C1);
     Y = tff(W);
     printgrsio(Y);
     fflush(stdout);

     YRS = tfrs(Y);
     printgrsio(YRS);
     fflush(stdout);
```

tfprint

```
        U = grstimesx(YRS);
        printgrsio(U);
        printivalio(RC = Omega(U));
        fflush(stdout);

        printivalio(BC = grseval(U,RC));
        fflush(stdout);
}
```

tfread

```
void
tfread()
{
        int i;


        W = readivalio();
        C1 = readivalio();
        Y = readgrsio();
        YRS = readgrsio();
        for(i=0; i <= YRS.n; ++i)YRS.f[i].k = 2;
        U = readgrsio();
        for(i=0; i <= U.n; ++i)U.f[i].k = 2;
        RC = readivalio();
        BC = readivalio();

}
```

r0

```
INTERVL
r0(w)
INTERVL w;
{
        INTERVL r, i1, cvtdi(), grseval(), w2, square();
        double rtest, rtest1, rtest2;
        int d;

        w2 = square(w);
        r = cvtbi(uplusb(U.f[0].center,negb(U.f[0].r)));
        i1 = rseval(U.f[0],r);
```

```
if(i1.up >= w2.dn){
        r.dn = (double) 0;
        d = -1;
        while(i1.dn < w2.up){
                ++d;
                i1 = U.f[d].p.p[0];
        }
        d = maxm(d,0);
        r.up = U.f[d].center.b;
        return(r);
}
r.up = RC.up;

for(;;){
        rtest = .5*(r.dn+r.up);
        i1 = grseval(U,cvtdi(rtest));
        if(i1.dn >= w2.up ) r.up = rtest;
        else if(i1.up <= w2.dn ) r.dn = rtest;
        else
        {
                rtest1 = rtest;
                d = 1;
                while(d ){
                        rtest2 = .5*(rtest+r.up);
                        if(rtest2 == rtest || rtest2 == r.up)d = 0;
                        i1 = grseval(U,cvtdi(rtest2));
                        if(i1.dn >= w2.up ) r.up = rtest2;
                        else rtest = rtest2;
                }

                rtest = rtest1;
                d = 1;
                while(d ){
                        rtest2 = .5*(rtest+r.dn);
                        if(rtest2 == rtest || rtest2 == r.dn)d = 0;
                        i1 = grseval(U,cvtdi(rtest2));
                        if(i1.up <= w2.dn ) r.dn = rtest2;
                        else rtest = rtest2;
                }
```

```
                if(r.dn <= r.up)return(r);
                else abort();
            }
        }
}

INTERVL
r1(w)
INTERVL w;
{
        INTERVL r, i1, cvtdi(), grseval(), w2, square();
        double rtest, rtest1, rtest2;
        int d;

        w2 = square(w);
        r = cvtbi(lplusb(U.f[U.n].center,U.f[U.n].r));
        i1 = rseval(U.f[U.n],r);
        if(i1.up >= w2.dn){
                r.up = 1.e15;
                d = U.n+1;
                while(i1.dn < w2.up){
                        --d;
                        i1 = U.f[d].p.p[0];
                }
                d = minm(d,U.n);
                r.dn = U.f[d].center.b;
                return(r);
        }
        r.dn = RC.dn;

        for(;;){
                rtest = 0.5*(r.up+r.dn);
                i1 = grseval(U,cvtdi(rtest));
                if(i1.dn >= w2.up ) r.dn = rtest;
                else if(i1.up <= w2.dn ) r.up = rtest;
                else {
                        rtest1 = rtest;
                        d = 1;
                        while( d ){
                                rtest2 = .5*(rtest+r.up);
```

r1

```
                        if(rtest2 == rtest || rtest2 == r.up) d = 0;
                        i1 = grseval(U,cvtdi(rtest2));
                        if(i1.up <= w2.dn ) r.up = rtest2;
                        else rtest = rtest2;
                }

                rtest = rtest1;
                d = 1;
                while(d ){
                        rtest2 = .5*(rtest+r.dn);
                        if(rtest2 == rtest || rtest2 == r.dn)d = 0;
                        i1 = grseval(U,cvtdi(rtest2));
                        if(i1.dn >= w2.up ) r.dn = rtest2;
                        else rtest = rtest2;
                }

                if(r.dn <= r.up)return(r);
                else abort();
        }
    }
}
```

vtfinf

```
vtfinf(a0, t, y0, y1)
double t, a0;
INTERVL *y0, *y1;
{
        RSERIES y, yp, ypp;
        INTERVL i1, ithreehalfs;
        BND eps, b0;
        double pnorm(), pw[SIZE], p[SIZE], alpha;
        int i, j, olddeg;
        alpha = (sqrt(73.0)−7)/2.0;

        olddeg = DEGREE;
        DEGREE = SIZE−1;
        pzer(p);
        p[0] = 1.0;
        p[1] = −a0*exp(log(t)*(−alpha));
```

```
eps.b = 1.0;
while(eps.b > 1.e-15){
        pcopy(p, pw);
        myprpower(p, 1.5, p);
        for(j=DEGREE; j >= 2; --j)
                p[j] = 12.0*p[j]/((3.0+j*alpha)*(4.0+j*alpha));
        p[0] = 1.0;
        p[1] = -a0*exp(log(t)*(-alpha));
        psub(p, pw, pw);
        eps.b = pnorm(pw);
}
y = rs(DEGREE,b0ero,bone);
for(i=2; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);
b0 = bl1nrs(y);
y.p.p[0] = ione;
y.p.p[1] = divi(cvtdi(-a0),iexp(mult(ilog(cvtdi(t)),ALPHA)));

if(grtb(b0,lcvtib(ration(3,10)))){
        printf("VTINF: condition 1 is WRONG\n");
        fflush(stdout);
        abort();
}
if(grtb(uabs(y.p.p[1]),lcvtib(ration(23,100)))){
        printf("VTINF: condition 2 is WRONG\n");
        fflush(stdout);
        abort();
}

ithreehalfs = divi(ithree,itwo);
yp = rspower(y,ithreehalfs);
ypp = rs(DEGREE, b0ero, bone);
for (i=2; i <= DEGREE; ++i){
        i1 = plus(ithree,mult(cvtinti(i),ALPHA));
        i1 = divi(mult(i1,plus(i1,ione)),cvtinti(12));
        ypp.p.p[i]= divi(yp.p.p[i],i1);
}
ypp.p.p[0]= y.p.p[0];
ypp.p.p[1]= y.p.p[1];
j = DEGREE+1;
i1 = plus(ithree,mult(cvtinti(j),ALPHA));
```

```
        i1 = divi(cvtinti(12),mult(i1,plus(i1,ione)));
        ypp.h = umultb(yp.h,ucvtib(i1));

        eps = bl1nrsf(rsminus(y,ypp));
        y.g = umultb(ucvtib(ration(75,10)),eps);
        *y0 = izero, *y1 = izero;
        for(i=0; i<=y.p.deg; ++i){
                *y0 = plus(*y0,y.p.p[i]);
                i1 = plus(ifour,mult(cvtinti(i),ALPHA));
                *y1 = plus(*y1,divi(yp.p.p[i],neg(i1)));
        }
        i1 = mult(cvtinti(144),iexp(mult(ilog(cvtdi(t)),neg(ithree))));
        *y0 = ienlarge(*y0,uplusb(y.g,y.h));
        *y0 = mult(*y0,i1);
        ypp = rspower(y,ithreehalfs);

        i1 = plus(ifour,mult(itwo,ALPHA));
        *y1 = ienlarge(*y1,uplusb(udivb(ypp.g,lcvtib(i1)),
            udivb(ypp.h,lplusb(bfour,lmultb(cvtintb(y.p.deg+1),
            lcvtib(ALPHA))))));
        i1 = mult(cvtinti(1728),iexp(mult(ilog(cvtdi(t)),neg(ifour))));
        *y1 = mult(*y1,i1);
        DEGREE = olddeg;
        freep(y.p), freep(yp.p), freep(ypp.p);
}
```

tfc1

```
tfc1(c)
double c;
{
        INTERVL i1, i2, i3, i4, i6, i5;
        double xin, eps;
        int i, j;

        i = Y.n−1;
        ;
        vtfinf(c, Y.f[i].center.b, &i5, &i6);
        i1 = i5;
        i2 = i6;
        for(j=i; j>=2; −−j){
```

```
		xin = Y.f[j].center.b;
		eps = supervtffx(xin, cvtdi(Y.f[j−1].center.b), i1, i2, &i1, &i2);

		i3 = Y.f[j−1].p.p[0];
		i4 = Y.f[j−1].p.p[1];
		if(i1.up <= i3.dn) return(−1);
		if(i2.dn >= i4.up) return(−1);
		if(i1.dn >= i3.up) return(1);
		if(i2.up <= i4.dn) return(1);
	}
	return(0);
}
```

getc1

```
getc1()
{
	double ctest;
	double t = 0.05;
	int k = 1;

	while(k ==1 || k == −1){
		printf("BOUNDS for C1\n");
		printival(C1);
		printivalio(C1);
		fflush(stdout);

		ctest = t*C1.up+(1.0−t)*C1.dn;
		k =  tfc1(ctest);
		if(k == 1) C1.dn = ctest;
		if(k == −1) C1.up = ctest;
	}
}
```

refiney

```
refiney()
{

	INTERVL i1, i2, i3, i4, i6, i5;
	INTERVL cvtdi(), intersect();
	double xin, eps;
	int i, j;
```

```
	i = Y.n;
	vtfinf(C1.up, Y.f[i].center.b, &i5, &i6);
	vtfinf(C1.dn, Y.f[i].center.b, &i3, &i4);
	i1.up =  i3.up;
	i1.dn =  i5.dn;
	i2.dn =  i4.dn;
	i2.up =  i6.up;
	i5 = Y.f[i].p.p[0];
	i6 = Y.f[i].p.p[1];
	Y.f[i].p.p[0] = intersect(i1,i5);
	Y.f[i].p.p[1] = intersect(i2,i6);
	for(j=i; j>=2; --j){
		xin = Y.f[j].center.b;
		eps = supervtffx(xin, cvtdi(Y.f[j-1].center.b), i1, i2, &i1, &i2);

		i3 = Y.f[j-1].p.p[0];
		i4 = Y.f[j-1].p.p[1];
		Y.f[j-1].p.p[0] = intersect(i1,i3);
		Y.f[j-1].p.p[1] = intersect(i2,i4);
		i1 = Y.f[j-1].p.p[0];
		i2 = Y.f[j-1].p.p[1];
	}
}
```

refine_numbers

```
refine_numbers()
{
	printivalio(W);
	printivalio(C1);
	refiney();
	printgrsio(Y);
	fflush(stdout);

	YRS = tfrs(Y);
	printgrsio(YRS);
	fflush(stdout);

	U = grstimesx(YRS);
	printgrsio(U);
	printivalio(RC = Omega(U));
```

```
	fflush(stdout);
	printivalio(BC = grseval(U,RC));
	fflush(stdout);
}
```

tfw2

```
tfw2(w)
double w;
{
	INTERVL i1, i2, i3, i4, i6, i5;
	INTERVL cvtdi();
	double xin, xout, eps;
	int j;

	vtff0(w, Y.f[1].center, &i5, &i6);
	i1 = i5;
	i2 = i6;
	for(j=1; j<=Y.n-1; ++j){
		xin = Y.f[j].center.b;
		xout = Y.f[j+1].center.b;

		eps = supervtffx(xin, cvtdi(xout), i1, i2, &i1, &i2);

		i3 = Y.f[j+1].p.p[0];
		i4 = Y.f[j+1].p.p[1];
		if(i1.up <= i3.dn) return(-1);
		if(i2.dn >= i4.up) return(1);
		if(i1.dn >= i3.up) return(1);
		if(i2.up <= i4.dn) return(-1);
	}
	return(0);
}
```

mygetw

```
mygetw()
{
	double wtest;
	double t = .05;
	int k = 1;

	while(k ==1 || k == -1){
```

```
                printf("BOUNDS for W\n");
                printival(W);
                printivalio(W);
                fflush(stdout);

                wtest = t*W.up + (1-t)*W.dn;
                k =  tfw2(wtest);
                if(k == 1) W.dn = wtest;
                if(k == -1) W.up = wtest;
        }
}
```

refineY

```
refineY()
{
        INTERVL i1, i2, i3, i4, i5, i6;
        int i;

        vtff0(W.up, Y.f[1].center, &i1, &i2);
        vtff0(W.dn, Y.f[1].center, &i5, &i6);
        Y.f[1].p.p[0] = iunion(i1,i5);
        Y.f[1].p.p[1] = iunion(i2,i6);
        for(i=1; i <= Y.n-1; ++i){
                i1 = Y.f[i].p.p[0];
                i2 = Y.f[i].p.p[1];
                supervtffx(Y.f[i].center.b, cvtbi(Y.f[i+1].center),
                    i1,i2,&i3,&i4);
                i1 = Y.f[i+1].p.p[0];
                i2 = Y.f[i+1].p.p[1];
                Y.f[i+1].p.p[0] = intersect(i1,i3);
                Y.f[i+1].p.p[1] = intersect(i2,i4);
        }

}
```

rstfu2

```
RSERIES
rstfu2(x,r,y)
INTERVL  x;
```

```
double r;
RSERIES  *y;
{
        int j;
        INTERVL u0, u1, a;
        RSERIES  u;


        a = plus(x,cvtdi(r));
        u0 = grseval(YRS,x);
        u1 = grsdereval(YRS,x);
        *y = vtffxi2( x , a, u0, u1);

        u = rs(y->p.deg, b0ero, y->r);
        for(j = y->p.deg; j>=1; --j)
                u.p.p[j] = plus(mult(y->p.p[j],x),
                    mult(y->p.p[j-1],cvtbi(y->r)));
        u.p.p[0] = mult(y->p.p[0] ,x);
        u.h = umultb(y->h,uplusb(uabs(x) ,y->r));
        u.h = uplusb(u.h,umultb(uabs(y->p.p[y->p.deg]),y->r));

        return(u);
}
```

yinf

```
RSERIES
yinf(t, m)
double t;
int m;
{
        RSERIES y, yp, ypp;
        INTERVL i1, ithreehalfs;
        BND eps, b0;
        double pnorm(), pw[SIZE], p[SIZE], alpha;
        int i, j, olddeg;
        alpha = (sqrt(73.0)-7)/2.0;
```

```
olddeg = DEGREE;
DEGREE = m;
pzer(p);
p[0] = 1.0;
p[1] = -(C1.up+C1.dn)*.5*exp(log(t)*(-alpha));
eps.b = 1.0;
while(eps.b > 1.e-15){
        pcopy(p, pw);
        myprpower(p, 1.5, p);
        for(j=DEGREE; j >= 2; --j)
                p[j] = 12.0*p[j]/((3.0+j*alpha)*(4.0+j*alpha));
        p[0] = 1.0;
        p[1] = -(C1.up+C1.dn)*.5*exp(log(t)*(-alpha));
        psub(p, pw, pw);
        eps.b = pnorm(pw);
}
y = rs(DEGREE,b0ero,bone);
for(i=2; i<= DEGREE; ++i) y.p.p[i] = cvtdi(p[i]);
b0 = bl1nrs(y);
y.p.p[0] = ione;
y.p.p[1] = divi(neg(C1),iexp(mult(ilog(cvtdi(t)),ALPHA)));

if(grtb(b0,lcvtib(ration(3,10)))){
        printf("YINF: condition 1 is WRONG\n");
        fflush(stdout);
        abort();
}
if(grtb(uabs(y.p.p[1]),lcvtib(ration(23,100)))){
        printf("YINF: condition 2 is WRONG\n");
        fflush(stdout);
        abort();
}

ithreehalfs = divi(ithree,itwo);
yp = rspower(y,ithreehalfs);
ypp = rs(DEGREE, b0ero, bone);
for (i=2; i <= DEGREE; ++i){
        i1 = plus(ithree,mult(cvtinti(i),ALPHA));
        i1 = divi(mult(i1,plus(i1,ione)),cvtinti(12));
        ypp.p.p[i]= divi(yp.p.p[i],i1);
```

```
	}
	ypp.p.p[0]= y.p.p[0];
	ypp.p.p[1]= y.p.p[1];
	j = DEGREE+1;
	i1 = plus(ithree,mult(cvtinti(j),ALPHA));
	i1 = divi(cvtinti(12),mult(i1,plus(i1,ione)));
	ypp.h = umultb(yp.h,ucvtib(i1));

	eps = bl1nrsf(rsminus(y,ypp));
	y.g = umultb(ucvtib(ration(75,10)),eps);
	y.k = 2;
	DEGREE = olddeg;
	freep(yp.p), freep(ypp.p);
	return(y);
}
```

expandY

```
GRS
expandY()
{
	GRS newy;
	double step;
	int i;
	INTERVL i1;

	step = Y.f[Y.n].center.b−Y.f[Y.n−1].center.b;
	if(step > 0.5) step = 0.5;
	i = (322.0−Y.f[Y.n].center.b)/step;

	newy = grs(i+Y.n);
	for(i=0; i <= Y.n; ++i){
		newy.f[i] = rscopy(Y.f[i]);
		freep(Y.f[n].p);
	}
	for(i=Y.n+1; i <= newy.n; ++i){
		newy.f[i] = rs(1, cvtdb(newy.f[i−1].center.b+step),bone);
		i1.up = zero;
		i1.dn = (double) −2;
		newy.f[i].p.p[1] = i1;
		i1.dn = zero;
```

```
            i1.up = one;
            newy.f[i].p.p[0] = i1;
        }

        free((char *)Y.f), Y.f = NULL;
        return(newy);
}
```