UNIVERSITÀ DEGLI STUDI DI BERGAMO
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
E METODI MATEMATICI°

QUADERNI DEL DIPARTIMENTO

Department of Information Technology and Mathematical Methods

Working Paper

Series "*Mathematics and Statistics*"

n. 7/MS – 2010

Some results concerning numerical approximations of Hecke's modular forms

by

Loïc Grenié, Giuseppe Molteni

° Viale Marconi. 5, I – 24044 Dalmine (BG), ITALY, Tel. +39-035-2052339; Fax. +39-035-562779

[§] L'accesso alle *Series* è approvato dal Comitato di Redazione. I *Working Papers* della Collana dei Quaderni del Dipartimento di Ingegneria dell'Informazione e Metodi Matematici costituiscono un servizio atto a fornire la tempestiva divulgazione dei risultati dell'attività di ricerca, siano essi in forma provvisoria o definitiva.

Some results concerning numerical approximations of Hecke's modular forms

using a technique of Köbe explained by Rudin

Loïc Grenié and Giuseppe Molteni

May 22, 2008

Contents

Note: we have appended the text of the two versions of the C-program we made to compute the terms of the series after the end of the text. The first version has an ad-hoc parallelization for single-core desktops and an MPI version. The second version has the improvement described in subsection 3.4 but has been reduced to the MPI version alone.

1 Introduction

We collect here the parts of [Hec] which are of interest for our work.

1.1 Hecke's modular forms

We let $\mathfrak{h}$ be the Poincaré half-plane of complex with positive imaginary part. We fix a real $\lambda > 0$. Let $\mathsf{G}(\lambda)$ be the group generated by the two actions on the complex projective plane

$$\begin{aligned} T &: z \longmapsto z + \lambda \\ S &: z \longmapsto -\frac{1}{z} \end{aligned}$$

and let $\mathsf{G}^*(\lambda)$ be the group generated by the three actions on the complex projective plane

$$\begin{aligned} \Sigma_1 &: z \longmapsto \frac{1}{\overline{z}} \\ \Sigma_2 &: z \longmapsto -\overline{z} \\ \Sigma_3 &: z \longmapsto -\overline{z} + \lambda \ . \end{aligned}$$

It is readily seen that $\mathsf{G}(\lambda)$ is a subgroup of index 2 of $\mathsf{G}^*(\lambda)$ and more precisely that an element $\prod_{i=1}^{\ell} \Sigma_{n_i}$ of $\mathsf{G}^*(\lambda)$ is an element of $\mathsf{G}(\lambda)$ if and only if ℓ is even.

Hecke [Hec] (see also [Ber]) proved that the action of $\mathsf{G}^*(\lambda)$ on $\mathfrak{h}$ is discrete if and only if $\lambda \geqslant 2$ or $\lambda = 2\cos(\pi/q)$ for $q \in \mathbf{N}_{\geqslant 3}$. For such values of λ, therefore, it is possible to introduce the class $\mathfrak{M}_0(\lambda, k, \varepsilon)$ of Hecke modular form of weight $k \in \mathbf{N}$ and multiplier $\varepsilon \in \{\pm 1\}$, i.e., the class of holomorphic functions $f : \mathfrak{h} \to \mathbf{C}$ having a representation as Fourier series

$$f(z) := \sum_{n=0}^{+\infty} a(n) \mathrm{e}^{2\pi i n z/\lambda} \tag{1}$$

with $a(n) \ll n^c$ for some constant $c > 0$, and satisfying

$$f(-1/z) = f(Sz) = f(\Sigma_1 \Sigma_2 z) = \varepsilon \left(\frac{i}{z}\right)^{-k} f(z).$$

Hecke shown that $\mathfrak{M}_0(\lambda, k, \varepsilon)$ is a finite dimensional $\mathbf{C}$-vector space if $\lambda \leqslant 2$ and produced formulas in terms of q, k, ε for such dimension. On the contrary, he also proved that $\mathfrak{M}_0(\lambda, k, \varepsilon)$ contains infinitely many $\mathbf{C}$-linearly independent forms when $\lambda > 2$. This is a remarkable fact if we consider that, for $\lambda = \sqrt{m}$ with $m \in \mathbf{N}$, the action of the classical congruence group $\Gamma_0(m)$ on $\mathfrak{h}$ is contained in $\mathsf{G}(\lambda)$ while the space of modular forms for $\Gamma_0(m)$ is finite-dimensional. The aim of the present work is to improve our comprehension of functions in $\mathfrak{M}_0(\lambda, k, \varepsilon)$ for $\lambda > 2$; for example, we would like to compare such functions with the classical modular forms associated with groups $\Gamma_0(m)$. In particular, it is known (see [Del, DS]) that the coefficients $a(n)$ of a (cusp new)-form associated with $\Gamma_0(m)$ and of weight k satisfy the bound $a(n) \ll_\epsilon n^{(k-1)/2+\epsilon}$; so we would like to answer to the following question: do the coefficients $a(n)$ of a function $f \in \mathfrak{M}_0(\lambda, k, \varepsilon)$ satisfy a similar bound? In our opinion the answer should be negative, essentially because the classical (new-cusp) forms support also a strong arithmetical structure (e.g., their coefficients are multiplicative) which should be absent in the general case (as it happens for $\lambda \leqslant 2$, see [KMP$^+$]), nevertheless the question is certainly worthy of study.

From now on we assume $\lambda > 2$.

1.2 Construction of Hecke's modular forms

The region

$$\mathcal{B}^* := \left\{ z \in \mathbf{C} : \ |z| > 1, 0 < \operatorname{Re} z < \frac{\lambda}{2} \right\}$$

is a fundamental domain for $\mathsf{G}^*(\lambda)$ and

$$\mathcal{B} := \mathcal{B}^* \cup \Sigma_2(\mathcal{B}^*) \cup \{\text{some imaginary}\} = \left\{ z \in \mathbf{C} : \ |z| > 1, |\operatorname{Re} z| < \frac{\lambda}{2} \right\}$$

is a fundamental domain for $\mathsf{G}(\lambda)$. A map in $\mathfrak{M}_0(\lambda, 0, 1)$ can be obtained in the following way. Let f be a conform map from $\mathcal{B}^*$ to the unit disk U (the existence and construction of such a map will be discussed later). The map f can be extended by continuity as injective map from $\overline{\mathcal{B}^*}$ into $\overline{U}$. Moreover, by the maximum modulus principle f maps $\partial\mathcal{B}^* \mapsto \partial U$, thus we get a new extension of f to the entire complex plain $\mathbf{C}$ by the Weierstrass' reflection principle, i.e., via the formulas

$$f(\Sigma_j z) = \frac{1}{\overline{f(z)}} \ , j = 1, 2, 3.$$

Note that such formulas imply that

$$f(Sz) = f(\Sigma_1\Sigma_2 z) = f(z) = f(\Sigma_3\Sigma_2 z) = f(Tz).$$

The map which we obtain in this way has poles at the points corresponding to the orbit (under the right coset $\mathsf{G}(\lambda)\Sigma_2$) of zeros of f and essential singularities at the limit points of such orbit; since $\mathfrak{h}$ is stable under $\mathsf{G}(\lambda)\Sigma_2$, the restriction of f to $\mathfrak{h}$ is holomorphic and bounded whenever the original map f has no zeros in $\mathcal{B}^* \cap \mathfrak{h}$.

Modifying a little bit an original argument of Hecke, it is possible to show how to employ the map f in order to generate maps in $\mathfrak{M}_0(\lambda, k, \varepsilon)$ for every choice of $k \in \mathbf{N}$ and every $\varepsilon \in \{\pm 1\}$. In fact, the point $z = i$ is the unique zero in $\overline{\mathcal{B}^*} \cap \mathfrak{h}$ of the map $g := f - f(i) \in \mathfrak{M}_0(\lambda, 0, 1)$, since f is injective here, by construction. Let $n_h(z_0)$ denote the order of a map h at $z = z_0$; then $n_g(i)$ is even, as a consequence of the general fact that for each map $h \in \mathfrak{M}_0(\lambda, k, \varepsilon)$ the equality

$$\varepsilon = (-1)^{n_h(i)}$$

holds (see [Ber, Cor. 5.4]). The map g is holomorphic in $\mathfrak{h}$, hence there exists a suitable open neighbourhood V of i where g assumes each value exactly $n_g(i)$ times (see [Ahl, Ch. 3, Th. 11]). V is covered by the four images of $\mathcal{B}^* \cap \mathfrak{h}$ under the group $\{\mathbf{1}, S, \Sigma_2, S\Sigma_2\}$ and g is injective in $\overline{\mathcal{B}^*} \cap \mathfrak{h}$, so that the bound $n_g(i) \leqslant 2$ follows. Since we already know that $n_g(i) \geqslant 2$, we conclude that $n_g(i) = 2$. This equality allows us to consider any analytic branch of the map $h := \sqrt{g}$ as an holomorphic map on $\mathfrak{h}$; this map has a zero of the first order at $z = i$ so that it is a map in $\mathfrak{M}_0(\lambda, 0, -1)$. The derivative of a map in $\mathfrak{M}_0(\lambda, 0, \varepsilon)$ is a map in $\mathfrak{M}_0(\lambda, 2, -\varepsilon)$, hence

$$f' \in \mathfrak{M}_0(\lambda, 2, -1), \quad \text{and} \quad h' = \frac{g'}{2\sqrt{g}} \in \mathfrak{M}_0(\lambda, 2, 1).$$

Note that h' has no zeros in $\mathfrak{h}$, since g is injective in $\overline{\mathcal{B}^*} \cap \mathfrak{h}$ and the zero of g' at $z = i$ (which is the unique one in $\overline{\mathcal{B}}$ because $g' = f'$) is cancelled in h' by the square root of g in its denominator. As a consequence there are two well defined analytic branches of $\sqrt{h'}$ and they belong to $\mathfrak{M}_0(\lambda, 1, 1)$, while $g\sqrt{h'}$ is a map in $\mathfrak{M}_0(\lambda, 1, -1)$. In this way we produced non-trivial maps in $\in \mathfrak{M}_0(\lambda, k, \varepsilon)$ for each combination of $k \in \{0, 1, 2\}$ and $\varepsilon \in \{-1, 1\}$; by taking suitable products of such maps we obtain maps also for every $k > 2$ and every ε.
Note: in the case $\lambda \leqslant 2$, Hecke proved that (a variation of) the construction we just shown produces the entire space $\mathfrak{M}_0(\lambda, k, \varepsilon)$. On the contrary, there is not any evidence that this happens also when $\lambda > 2$, but we think (hope) that the maps we have constructed following the previous method are sufficiently near the generic map in Hecke's spaces and that our maps give a good insight of the generic behavior of the elements of $\mathfrak{M}_0(\lambda, k, \varepsilon)$.

As we see, the previous construction relays upon our ability to produce a conformal map $\mathcal{B}^* \mapsto U$. The existence of such a map is assured by the Riemann's theorem. The classical proof of that theorem is not constructive (for example, see [Rud, Ch. 14]), nevertheless the first general and complete proof of such theorem was given by Köbe (see [Köb]) and was quite explicit. In next section we modify the construction of Köbe in order to obtain a procedure that can be implemented as a routine on a computer. In this way we will be able to analyze at least numerically the map.

Note: the Fourier coefficients $b(n)$ of the functions in spaces $\mathfrak{M}_0(\lambda, k, \varepsilon)$ we produce according to the previous method from the original map $f \in \mathfrak{M}_0(\lambda, 0, 1)$ will satisfy the Ramanujan bound $b(n) \ll_\varepsilon n^{(k-1)/2+\varepsilon}$, whenever the Fourier coefficients $a(n)$ of f will satisfy the Ramanujan bound $a(n) \ll_\varepsilon n^{-1/2+\varepsilon}$. Thus, our attention will be mainly payed to the Fourier coefficients of such map.

1.3 (semi-)trivial bound

The following argument shows that the Fourier coefficients in (1) are certainly $o(1)$: this claim is a weak but interesting improvement on the bound $a(n) = O(1)$ which is the standard bound usually proved for such coefficients.

Let $f : \mathfrak{h} \to \mathbf{C}$ be a holomorphic, bounded and Σ_3-invariant function, so that the Fourier representation

$$f(z) := \sum_{n=0}^{+\infty} a(n) \mathrm{e}^{2\pi i n z/\lambda}$$

holds for $z \in \mathfrak{h}$. By hypothesis $\|f\|_{\mathfrak{h},\infty} < +\infty$, therefore by the Parseval's identity we get

$$\sum_{n=0}^{+\infty} |a(n)|^2 \mathrm{e}^{-4\pi n y/\lambda} = \frac{1}{\lambda} \int_0^\lambda |f(x+iy)|^2 dx \leqslant \|f\|^2_{\mathfrak{h},\infty},$$

for every $y > 0$. Let $N \in \mathbf{N}$ be fixed; setting $y = \lambda/N$ in previous inequality we get

$$\mathrm{e}^{-4\pi} \sum_{n \leqslant N} |a(n)|^2 \leqslant \sum_{n \leqslant N} |a(n)|^2 \mathrm{e}^{-4\pi n/N} \leqslant \sum_{n=0}^{+\infty} |a(n)|^2 \mathrm{e}^{-4\pi n/N} \leqslant \|f\|^2_{\mathfrak{h},\infty},$$

implying that $\sum_{n \leqslant N} |a(n)|^2 \leqslant \mathrm{e}^{4\pi} \|f\|^2_{\mathfrak{h},\infty}$. Since this bound is independent on N we obtain that $\sum_{n=0}^{+\infty} |a(n)|^2 \leqslant \mathrm{e}^{4\pi} \|f\|^2_{\mathfrak{h},\infty}$ so that $a(n) \in \ell^2(\mathbf{C})$ and, in particular,

$$a(n) \to 0 \quad \text{as} \quad n \to \infty.$$

2 Köbe's technique

2.1 The f_α maps

Let U denote the open unit disc in the complex plane and let α and u be in U. We let

$$\varphi_\alpha(u) := \frac{u - \alpha}{1 - \overline{\alpha} u}.$$

The set of $\zeta\varphi_\alpha$, for $|\zeta| = 1$, is the set of conformal mappings between U and itself. We can extend φ_α to $\overline{U}$ by the same formula for $u \in \overline{U}$ and it remains a bijection between $\overline{U}$ and itself. Observe that $\varphi_\alpha(0) = -\alpha$ and $\varphi_\alpha(\alpha) = 0$.

Let Ω be a simply connected open properly contained in U and containing 0. Let α be in $U \backslash \Omega$ and β be one of the two square roots of $-\alpha$. We put $w(u) := u^2$. Consider

$$F_\alpha := \varphi_{-\alpha} \circ w \circ \varphi_{-\beta}.$$

Let $V := F_\alpha^{-1}(\Omega)$. The open set V has two connected components V_0 and V_1, with $\beta \in V_0$ and $-\beta \in V_1$ and the restriction of F_α to either V_i is injective. Indeed $F_\alpha^{-1}(\Omega) = \varphi_\beta \circ w^{-1} \circ \varphi_\alpha(\Omega)$; but φ_α and φ_β are bijective and analytic so that they conserve the number of connected components; now $\varphi_\alpha(\Omega)$ is simply connected and does not contain 0 (because $\varphi_\alpha(\alpha) = 0$, φ_α is bijective and $\alpha \notin \Omega$), so that $w^{-1}(\varphi_\alpha^{-1})$ has two connected components.

Let G_α be the inverse of F_α as map between V_0 and Ω and let $f_\alpha : \Omega \to V_0$ with $f_\alpha(z) := \lambda_\alpha G_\alpha(z)$ and $\lambda_\alpha = \frac{|G'_\alpha(0)|}{G'_\alpha(0)}$. Observe that $|\lambda_\alpha| = 1$ and that f_α does not depend on the choice of β (while F_α and G_α do).

2.2 Properties of f_α

Proposition 1 *The application $f_\alpha : \Omega \to V_0$ is injective, $f_\alpha(0) = 0$, $f_\alpha(\alpha) = \frac{\alpha}{\sqrt{|\alpha|}}$ and*

$$f'_\alpha(0) = |G'_\alpha(0)| = \frac{1+|\alpha|}{2\sqrt{|\alpha|}} \tag{2}$$

Proof: The three first assertions are immediate from the construction. We have

$$G'_\alpha(0) = \frac{1}{F'_\alpha(0)} = \frac{1}{\varphi'_{-\beta}(0)2\varphi_{-\beta}(0)\varphi'_{-\alpha}(\varphi_{-\beta}(0)^2)}.$$

Since

$$\varphi'_\alpha(u) = \frac{1}{1-\overline{\alpha}u} + \overline{\alpha}\frac{u-\alpha}{(1-\overline{\alpha}u)^2} = \frac{1-|\alpha|^2}{(1-\overline{\alpha}u)^2}$$

it follows

$$G'_\alpha(0) = \frac{1}{2(1-|\alpha|)\beta\frac{1-|\alpha|^2}{(1-|\alpha|^2)^2}} = \frac{1+|\alpha|}{2\beta} \tag{3}$$

and the result follows. □

Proposition 2 *We have*

$$\forall u \in \Omega, \quad u \neq 0 \Rightarrow |f_\alpha(u)| > |u|.$$

Proof: Since f_α is, up to a complex sign, the inverse of F_α, this will follow immediately from

$$\forall u \in U, \quad u \neq 0 \Rightarrow |F_\alpha(u)| < |u|.$$

Let $u \in U$, $u \neq 0$; recall that $\alpha = -\beta^2$. We have

$$\begin{aligned} F_\alpha(u) = \varphi_{-\alpha} \circ w \circ \varphi_{-\beta}(u) &= \frac{\left(\frac{u+\beta}{1+\overline{\beta}u}\right)^2 + \alpha}{1+\overline{\alpha}\left(\frac{u+\beta}{1+\overline{\beta}u}\right)^2} \\ &= \frac{(u+\beta)^2 - \beta^2(1+\overline{\beta}u)^2}{(1+\overline{\beta}u)^2 - \overline{\beta}^2(u+\beta)^2} \\ &= \frac{(1-|\beta|^4)u^2 + 2\beta(1-|\beta|^2)u}{(1-|\beta|^4) \quad + 2\overline{\beta}(1-|\beta|^2)u} \\ &= u\frac{u+\frac{2\beta}{1+|\beta|}}{1+\frac{2\overline{\beta}}{1+|\beta|^2}u} = u\varphi_{-\gamma}(u) \end{aligned}$$

where $\gamma = \frac{2\beta}{1+|\beta|^2}$ is an element of U. We know that $\varphi_{-\gamma}(u) \in U$, thus $|F_\alpha(u)| < |u|$. □

Proposition 3 *Recall that $f_\alpha : \Omega \to V_0$. We have*

$$\forall z \in \Omega, \quad f_\alpha(z) = s\left(|\beta| + \frac{1}{|\beta|}\right)^{-1}\left(1 + \frac{z}{s} - \sqrt{\left(1 - \frac{|\alpha| z}{s}\right)\left(1 - \frac{z}{s|\alpha|}\right)}\right)$$

where $s = \frac{\alpha}{|\alpha|}$ is the complex sign of α and the branch of the square root is the one such that $f_\alpha(0) = 0$, that is such that $\sqrt{1} = 1$.

In particular, if $z = \alpha + its$, $t \in \mathbf{R}$, then

$$f_\alpha(z) = \frac{\alpha}{\sqrt{|\alpha|}} - \sqrt{|t|}\sqrt{\frac{1-|\alpha|}{1+|\alpha|}}\; s\left(\frac{1-i}{\sqrt{2}}\right)^{\operatorname{sgn} t} + O(t) \quad \text{as } t \to 0.$$

Proof: Let's call h_α the expression given in the text of the proposition. Our choice of the square root shows that $h_\alpha(0) = 0 = f_\alpha(0)$. From (3) we see that $\lambda_\alpha = \frac{\beta}{|\beta|}$, so that $s = -\lambda_\alpha^2$. We thus compute

$$\begin{aligned}
\frac{h_\alpha(z)}{\lambda_\alpha} &= -\lambda_\alpha\left(|\beta| + \frac{1}{|\beta|}\right)^{-1}\left(1 + \frac{z}{s} - \sqrt{\left(1 - \frac{|\alpha| z}{s}\right)\left(1 - \frac{z}{s|\alpha|}\right)}\right) \\
&= -\frac{\beta}{|\alpha| + 1}\left(1 + \frac{z}{s} - \sqrt{(1 - \overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}\right).
\end{aligned}$$

Then

$$\begin{aligned}
\varphi_{-\beta}\left(\frac{1}{\lambda_\alpha} h_\alpha(z)\right) &= \frac{-\frac{\beta}{|\alpha|+1}\left(1 + \frac{z}{s} - \sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}\right) + \beta}{1 - \overline{\beta}\frac{\beta}{|\alpha|+1}\left(1 + \frac{z}{s} - \sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}\right)} \\
&= \beta\frac{|\alpha| + 1 - 1 - \frac{z}{s} + \sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}}{|\alpha| + 1 - |\beta|^2\left(1 + \frac{z}{s} - \sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}\right)} \\
&= \beta\frac{|\alpha| - \frac{z}{s} + \sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}}{1 - |\alpha|\frac{z}{s} + |\alpha|\sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}} \\
&= \frac{\beta}{|\alpha|}\cdot\frac{|\alpha|^2 - \overline{\alpha} z + |\alpha|\sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}}{1 - \overline{\alpha} z + |\alpha|\sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}} \\
&= \beta\frac{\left(1 - \overline{\alpha} z - |\alpha|^2 + \overline{\alpha} z\right)\sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}}{(1-\overline{\alpha} z)^2 - |\alpha|^2(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)} \\
&= \beta\frac{(1 - |\alpha|^2)\sqrt{(1-\overline{\alpha} z)\left(1 - \frac{z}{\alpha}\right)}}{1 - 2\overline{\alpha} z + \overline{\alpha}^2 z^2 - |\alpha|^2 + (1 + |\alpha|^2)\overline{\alpha} z - \overline{\alpha}^2 z^2}
\end{aligned}$$

$$= \beta \frac{\sqrt{(1-\overline{\alpha}z)\left(1-\frac{z}{\alpha}\right)}}{1-\overline{\alpha}z}$$

It follows

$$\left[\varphi_{-\beta}\left(\frac{1}{t}h_\alpha(z)\right)\right]^2 = -\alpha\frac{(1-\overline{\alpha}z)\left(1-\frac{z}{\alpha}\right)}{(1-\overline{\alpha}z)^2} = \varphi_\alpha(z)$$

□

Algorithmically this function is better than the one we get directly from the proof because, apart from products by constants and sums, it only has one polynomial of degree 2 and one square root to compute. When one uses a computational program, the square root is usually the principal square root, i.e. the one whose values have argument in $\left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$. In order to have an analytical square root function, in that case we need to change the square root when we cross the negative real half-axis. The polynomial is a negative real when z is on the segment going from α to $\frac{1}{\overline{\alpha}}$ and on its perpendicular bisector (which does not meet the unit disc). We see that the probability of needing to change the determination of the square root during a computation is low: it happens only if z crosses the segment going from α to $\frac{\alpha}{|\alpha|}$. In the computer program we realized, we chose to ignore such a possibility.

Proposition 4 *The map f_α can be extended as homeomorphism from $\overline{\Omega}$ onto $\overline{V_0}$. Moreover, if $\partial U \cap \partial\Omega \neq \emptyset$, then $|f_\alpha(z)| = 1$ for every $z \in \partial U \cap \partial\Omega$.*

Proof: Recalling that $F_\alpha := \varphi_{-\alpha} \circ w \circ \varphi_{-\beta}$, it is immediate to verify that F_α can be extended as a continuous bijection from $\overline{V_0}$ onto $\overline{\Omega}$. Being $\overline{V_0}$ a compact set, $F_\alpha(K)$ is closed whenever $K \subseteq \overline{V_0}$ is closed, therefore the inverse map $G_\alpha : \overline{\Omega} \to \overline{V_0}$ is continuous. Since $f_\alpha := \lambda_\alpha G_\alpha$ with $\lambda_0 := \frac{|G'_\alpha(0)|}{G'_\alpha(0)}$, also f_α is a homeomorphism $\overline{\Omega} \to \overline{V_0}$.

For the second claim we remark that $|F_\alpha(u)| = 1$ if and only if $|u| = 1$. As a consequence, if $|z| = 1$ and $z \in \overline{\Omega}$, then $|F_\alpha(G_\alpha(z))| = |z| = 1$ so that $|f_\alpha(z)| = |G_\alpha(z)| = 1$. □

2.3 The iteration process

We recall some definitions from [SS, Ch. 1, ¶3].

Definition 5

- *A **closed parametrized curve** is a continuous application $z : S^1 \to \mathbf{C}$.*
- *A closed parametrized curve z is said **simple** if z is injective.*
- *A closed parametrized curve z is said **smooth** if $z'(t)$ exists for all $t \in S^1$ and z' is continuous.*
- *A closed parametrized curve z is said **piecewise-smooth** if there exist $\{a_i\}_{i=1}^n \subset (0;1)$ such that $t \mapsto z(\mathrm{e}^{2\pi i t})$ is smooth on $(a_i; a_{i+1})$ and $(a_n; 1 + a_1)$.*

– *A **closed curve**, **simple closed curve**, **smooth closed curve** and **piecewise-smooth closed curve** is the image of the corresponding closed parametrized curves.*

Theorem 6 *Let Ω be a simply connected open properly contained in U and containing 0. Suppose that $\partial\Omega$ is a simple, piecewise-smooth closed curve. Let $\{x_n\}$ be a sequence in $\partial\Omega \setminus \partial U$ dense in $\partial\Omega \setminus \partial U$. We define sequences of elements α_n of U and of applications $f_n : \overline{\Omega} \to \overline{U}$ in the following way*

1) Let α_0 be an arbitrary element of U and let $f_0 := \mathrm{id}$.

2) For each $n \geqslant 0$ such that f_n has been defined, we let $\alpha_{n+1} := f_n(x_n)$ and $f_{n+1} := f_{\alpha_{n+1}} \circ f_n$ (for any choice of β in Section 2.1).

Then the sequence $\{f_n\}$ converges uniformly on each compact of Ω to a conformal bijection f from Ω onto U that can be extended to a continuous bijection from $\overline{\Omega}$ onto $\overline{U}$.

Remark 7

1. Assumption about the smoothness of $\partial\Omega$ allows us to assure that the map which we will define in Ω will be extendible in continuous way to $\overline{\Omega}$.
2. The map f_α is well defined only if $\alpha \in U$; for $\alpha \in \partial U$, in fact, $\varphi_\alpha(u) = -\alpha$ for each u and G_α does not exist. Assumption $x_n \in \partial\Omega \setminus \partial U$ and definition $\alpha_{n+1} := f_n(x_n)$ in previous theorem assure that $\alpha_n \in U$ for every n.

Proof:

a) Note that $f_n(0) = 0$ for each n; let h_n be the holomorphic function on Ω such that $f_n(u) = uh_n(u)$. Then $|h_{n+1}(u)| > |h_n(u)|$ for each $u \in \Omega$.

   This follows immediately from the fact, seen in Proposition 2, that $|f_\alpha(u)| > |u|$ (for $u \neq 0$) and $h_n(0) = f_n'(0)$ so that $h_{n+1}(0) = \frac{1+|\alpha_{n+1}|}{2\sqrt{|\alpha_{n+1}|}} h_n(0)$.

b) We know that f_n is injective and that $f_n(0) = 0$ (and thus $f_n'(0) \neq 0$) hence h_n is everywhere non-zero. We claim that the sequence $\{\ln |h_n|\}$ converges uniformly on every compact of Ω.

   The claim will follow from the Harnack's theorem [Rud, Th. 11.11, pg. 236] if we will be able to prove that $\ln |h_n(0)|$ converges to $\ln \lim f_n'(0)$. Let $r_0 := \sup\{r : D(0,r) \subset \Omega\}$; the map $u \mapsto f_n(r_0 u)$ is injective from U to U, and thus, from Schwartz's lemma, we get that $|r_0 f_n'(0)| \leqslant 1$, proving that the sequence $\{f_n'(0)\}$ is bounded. Moreover, by induction we see that

$$f_n'(0) = \prod_{k=0}^{n} \frac{1+|\alpha_k|}{2\sqrt{|\alpha_k|}};$$

   this identity shows that $f_n'(0)$ increases monotonously as $n \to \infty$ (because $\frac{1+|\alpha_n|}{2\sqrt{|\alpha_n|}} \geqslant 1$), completing the proof that $f_n'(0)$ converges to a positive constant. As a byproduct we get that $\frac{1+|\alpha_n|}{2\sqrt{|\alpha_n|}} = \frac{f_n'(0)}{f_{n-1}'(0)} \to 1$, so that $\lim |\alpha_n| = 1$.

c) The sequence f_n converges uniformly on every compact of Ω.

We know that h_n is everywhere non-zero. Since Ω is simply connected, there is a logarithm of h_n. Since $h_n(0) > 0$, we choose the logarithm such that $\ln h_n(0) \in \mathbf{R}$ and by b) the sequence $\{\ln h_n(0)\}$ converges. We thus know that $\ln h_n$ converges uniformly on every compact of Ω. It follows that h_n converges as well on compact subsets. Thus f_n converges uniformly on every compact of Ω towards a function f.

d) $f : \Omega \to \overline{U}$ is a holomorphic non-constant map, in particular $f(\Omega) \subseteq U$.

In fact, the uniform convergence on every compact set in U of the sequence of holomorphic maps f_n implies that f is holomorphic as well. Moreover, by the Cauchy integral representation it follows that also their derivatives converge uniformly on compact sets to f'; in particular we have

$$f'(0) = \lim_{n \to +\infty} f_n'(0) = \prod_{k=0}^{+\infty} \frac{1 + |\alpha_k|}{2\sqrt{|\alpha_k|}} \neq 0,$$

proving that f is not constant. In particular, the inclusion $f(\Omega) \subseteq U$ follows from the maximum modulus principle.

e) $f : \Omega \to U$ is injective.

Each f_n is injective and we know that f is not constant, thus the claim follows from [SS, Ch. 8, Prop. 3.5].

f) f can be continuously extended as an injective map from $\overline{\Omega}$ onto $f(\overline{\Omega})$.

We just proved that $f : \Omega \to f(\Omega)$ is a conformal bijection and by hypothesis $\partial\Omega$ is a simple piecewise-smooth close curve; thus the claim follows by an immediate generalization of the argument in [SS, Ch. 8, Th. 4.2] (see also [SS, Ch. 8, ex. 18 pg. 252], observing that the hypothesis "simple" is necessary but erroneously omitted in ex. 18).

g) $f\big((\partial\Omega) \cap (\partial U)\big) \subseteq \partial U$.

Recalling that $f_0 = \mathrm{id}$ and that $f_{n+1} = f_{\alpha_{n+1}} \circ f_n$ and using Prop. 4 it is easy to prove by induction on n that $|x| = 1$ implies $|f_n(x)| = 1$ for every n. As a consequence, taking the limit in n we get $|f(x)| = 1$ for every $x \in (\partial\Omega) \cap (\partial U)$.

h) $f(\partial\Omega) \subseteq \partial U$.

We know that $f : \overline{\Omega} \to f(\overline{\Omega}) \subseteq \overline{U}$ is continuous and that $|f_{\alpha_{n+1}}(u)| \geqslant |u|$, for any $u \in f_n(\Omega)$, hence for any $u \in \Omega$ the sequence $|f_n(u)|$ is increasing; continuity implies that the same fact holds also for every $u \in \overline{\Omega}$. As a consequence, we have

$$|f_n(x_n)| \leq |f_m(x_n)| \leq 1$$

for each $m \geq n$ and taking the limit $m \to \infty$ we get

$$|f_n(x_n)| \leq |f(x_n)| \leq 1. \tag{4}$$

We have already seen that in g) that $\forall z \in (\partial\Omega) \cap (\partial U)$, we have $f(z) \in \partial U$. Fix $z \in \partial\Omega \backslash \partial U$ and let $\{x_{n_k}\}$ be a subsequence of $\{x_n\}$ such that $x_{n_k} \to z$ as $k \to \infty$. From (4) it follows

$$|f_{n_k}(x_{n_k})| \leq |f(x_{n_k})| \leq 1. \tag{5}$$

In b) we proved that $\lim |\alpha_n| = \lim |f_n(x_n)| = 1$, hence from (5) we get $|f(z)| = 1$.

i) $f(\Omega) = U$.

By h) we know that $f(\partial\Omega) \subseteq \partial U$. Moreover, we know that $f(\Omega) \subseteq U$, that f is a conformal map from Ω to $f(\Omega)$ and that it is a continuous bijection from $\overline{\Omega}$ to $f(\overline{\Omega})$: the claim follows from these facts. In fact, under these hypotheses $f(\partial\Omega)$ coincides with ∂U: suppose that $f(\partial\Omega) \neq \partial U$, then $f(\partial\Omega)$ is homeomorphic to a segment (since $\partial\Omega$ is connected). Let p be a point in $\partial\Omega$. Then $\partial\Omega \setminus \{p\}$ is still connected, while $f(\partial\Omega \setminus \{p\}) = f(\partial\Omega) \setminus \{f(p)\}$ is disconnected.

Let $q \in U$. Let d be the distance of q to ∂U. Since $f(\partial\Omega) = \partial U$, the uniform continuity of f in $\overline{\Omega}$ allows us to find a smooth simple curve γ in Ω near enough to $\partial\Omega$ in such a way that $f(\gamma)$ is a smooth curve in U whose distance to ∂U is $< d$. In particular, $f(\gamma)$ wraps around q, thus

$$\pm 2\pi i = \int_{f(\gamma)} \frac{dz}{z - q} = \int_\gamma \frac{f'(w)}{f(w) - q}\, dw.$$

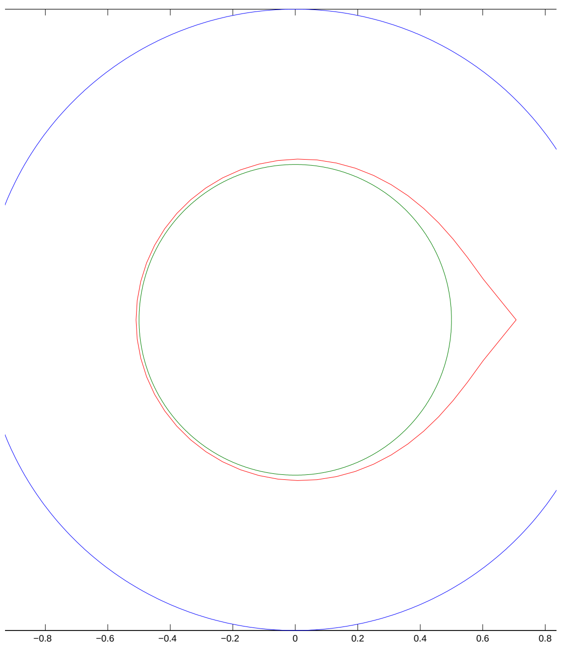
This formula shows that $f(w) = q$ has a solution in the part of Ω encircled by γ.

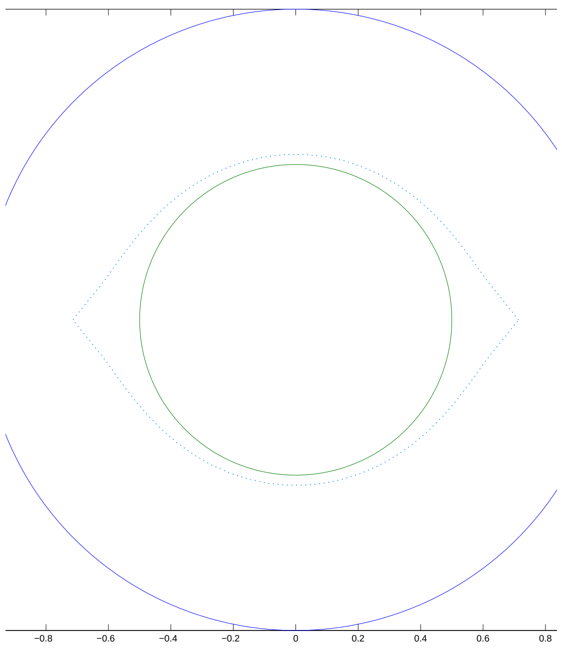
□

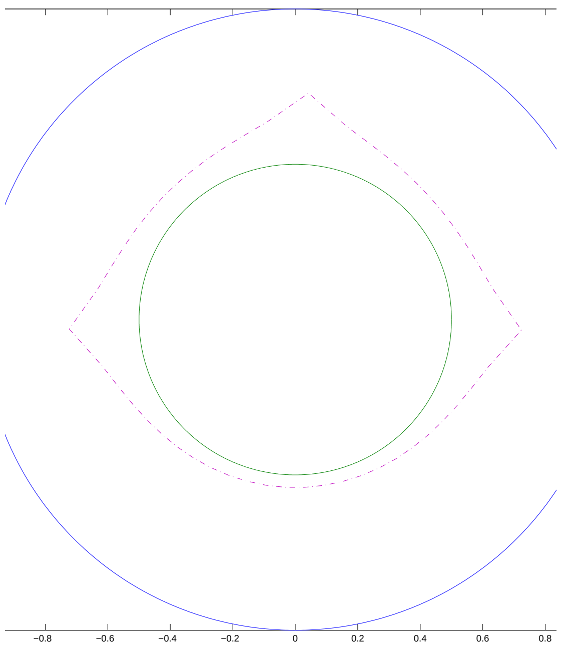
Remark 8 Observe that any sequence of $\{x_n\}$ appearing in Theorem 6 leads to a conformal bijection f between Ω and U such that $f(0) = 0$ and $f'(0) > 0$. If f_1 and f_2 are two of them, then $f_1 \circ f_2^{-1}$ is a conformal bijection between U and itself. This means that $f_1 \circ f_2^{-1} = \zeta\varphi_\alpha$ for some $\alpha \in U$ and some $\zeta \in S^1$. We have $\varphi_\alpha(0) = f_1(f_2^{-1}(0)) = f_1(0) = 0$ so that $\alpha = 0$ and $\zeta = \frac{f_1'(0)}{f_2'(0)} > 0$ so that $\zeta = 1$. We thus have $f_1 = f_2$, which means that the choice of the sequence of points on the border is irrelevant (as long as it is dense).

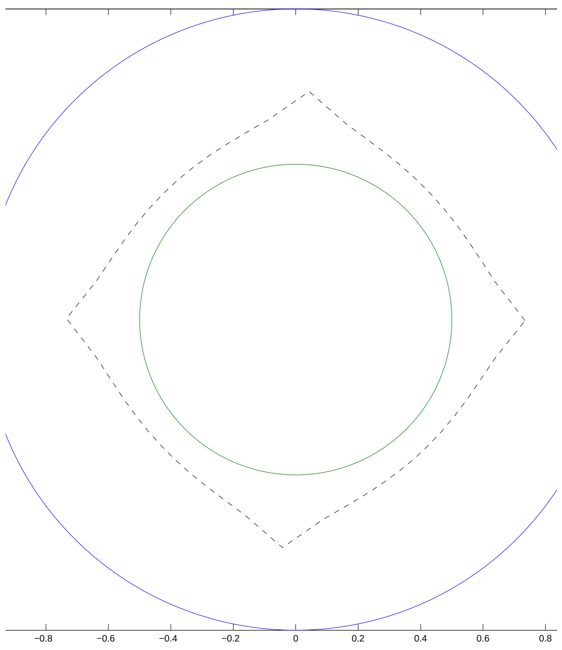
2.4 Graphical description

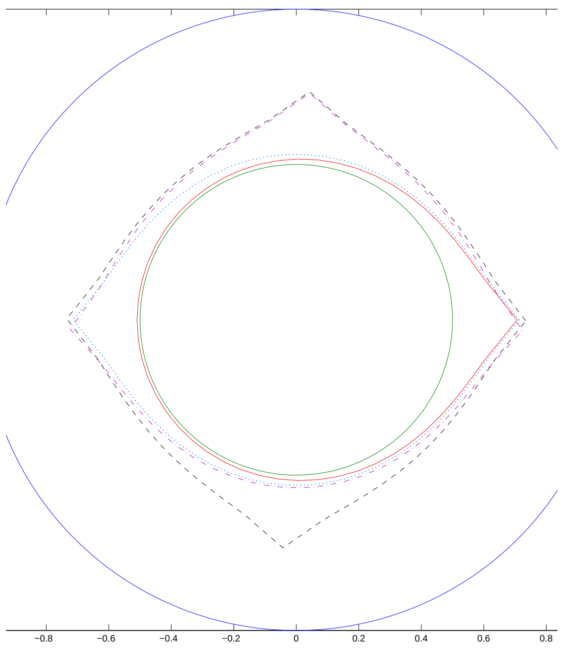
We describe the method in the following diagrams. The base set Ω is the disc of radius $\frac{1}{2}$. The first four graphics show this set along with its four first images using Köbe's method. The last one represents all five graphs together.


−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8


−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8


−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8


−0.8
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
0.8



3 Gluing them together

We explain how to combine Hecke's technique with Köbe's technique to obtain a Hecke modular form of weight 0.

3.1 The three complex planes

We will use three complex planes. They will be identified by their variables: z, q and u, respectively. The maps are the following:

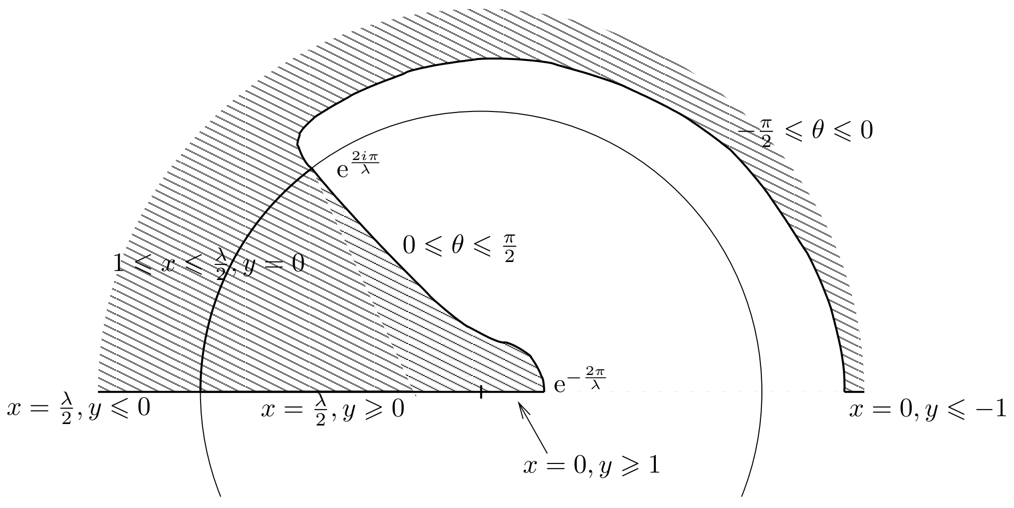
$$q = \mathrm{e}^{\frac{2i\pi}{\lambda}z}$$
$$u = \zeta\frac{t-q}{\bar{t}-q}$$
$$q = \frac{t\zeta - \bar{t}u}{\zeta - u}$$

The parameter ζ is a complex of modulus 1 and serves to adjust the mapping to be nice. The parameter t will be fixed below.

In the z plane, we have Poincaré's half-plane $\mathfrak{h}$ while in the q and u planes we have the unitary discs U_q and U. Observe that $q(\mathfrak{h}) = U_q \backslash \{0\}$ and that $q(z_1) = q(z_2) \iff z_1 - z_2 \in \lambda\mathbf{Z}$. The maps $q(u)$ and $u(q)$ are diffeomorphisms between $\mathbf{C}_u \backslash \{\zeta\}$ and $\mathbf{C}_q \backslash \{\,\bar{t}\,\}$.

3.2 The domain

Let Ω_q be the image of $\Omega_z = \mathcal{B}^*$ under the application $z \mapsto q(z)$.


$-\frac{\pi}{2} \leqslant \theta \leqslant 0$
$e^{\frac{2i\pi}{\lambda}}$
$0 \leqslant \theta \leqslant \frac{\pi}{2}$
$1 \leqslant x \leqslant \frac{\lambda}{2}, y = 0$
$e^{-\frac{2\pi}{\lambda}}$
$x = \frac{\lambda}{2}, y \leqslant 0$
$x = \frac{\lambda}{2}, y \geqslant 0$
$x = 0, y \leqslant -1$
$x = 0, y \geqslant 1$

We have tried to represent the domain Ω_q. The image of $q(\mathcal{B}^* \cap \mathfrak{h})$ is the portion inside the unitary disc. We have tried to label the borders of the image with the corresponding coordinates in the z plane. The curve delimiting $q(\mathcal{B}^* \cap \mathfrak{h})$ has a flex if and only if $\lambda < 2\pi$ (in the point corresponding to $z = e^{i \arcsin \frac{\lambda}{2\pi}}$).

We use a parameter $t \in i\mathbf{R}$ such that $t \in \Omega_q$ and take $\zeta = -1$. We consider $\Omega = \Omega_u$ the image of Ω_q under the map $q \mapsto u(q) = \frac{t-q}{t+q}$. The open Ω is included in U, contains 0 and is unequal to U (it lacks the image of the half disc excluded from $\mathcal{B}^*$).

3.3 Improving the domain

The border of Ω_q has two singularities at the points $m = e^{-\frac{2\pi}{\lambda}}$ and $M = e^{\frac{2\pi}{\lambda}}$. This means that the derivative of Köbe's function will vanish at those two points. Unfortunately, these singularities slow down the rate of convergence of the algorithm. To improve the situation, we consider the function $h(q) = \frac{q^2-1}{q-c}$ where $c = \cosh(\frac{2\pi}{\lambda})$. Note that $h(m) = 2m$ and $h(M) = 2M$. Denote x and y the real and imaginary parts of q and $s = \sinh(\frac{2\pi}{\lambda})$. One sees immediately that $\operatorname{Im} h(q) = \frac{(x-c)^2+y^2-s^2}{(x-c)^2+y^2} y$. This means that if $\operatorname{Im} q = y > 0$ then $\operatorname{Im} h(q) > 0$ if and only if q lies outside the circle of center c and radius s, which is also the circle of diameter $[m, M]$. Since $h'(q) = \frac{q^2-2qc+1}{(q-c)^2} = \frac{(q-m)(q-M)}{(q-c)^2}$, we see that $h'(m) = h'(M) = 0$. We thus take $\Omega'_q = h(\Omega_q)$ and apply Köbe's method to Ω'_q instead of Ω_q. Indeed, since $h'(m) = h'(M) = 0$, the curve delimiting Ω'_q does not have a singularity, and since Ω_q lies outside the circle of radius $[m, M]$ (see Lemma 9 below), the elements of Ω'_q all have positive imaginary part. We thus take $u(q) := \zeta\frac{t-h(q)}{\bar{t}-h(q)}$ for any t such that $t \in \Omega'_q$. For definiteness, we choose $t = h(q(2))$ (which has the side effect that we take

$\lambda > 4$). We then have

$$u(h(q)) = \zeta \frac{\frac{\mathrm{e}^{\frac{8i\pi}{\lambda}}-1}{\left(\mathrm{e}^{\frac{4i\pi}{\lambda}}-c\right)} - \frac{q^2-1}{q-c}}{\frac{\mathrm{e}^{-\frac{8i\pi}{\lambda}}-1}{\left(\mathrm{e}^{-\frac{4i\pi}{\lambda}}-c\right)} - \frac{q^2-1}{q-c}} = \zeta \frac{\frac{\mathrm{e}^{\frac{4i\pi}{\lambda}}-\mathrm{e}^{-\frac{4i\pi}{\lambda}}}{\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right)} - \frac{q^2-1}{q-c}}{\frac{\mathrm{e}^{-\frac{4i\pi}{\lambda}}-\mathrm{e}^{\frac{4i\pi}{\lambda}}}{\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right)} - \frac{q^2-1}{q-c}}$$

$$= \zeta \frac{1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}}{1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}} \cdot \frac{(q-c)i\sin\frac{4\pi}{\lambda} - \frac{q^2-1}{2}\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right)}{-(q-c)i\sin\frac{4\pi}{\lambda} - \frac{q^2-1}{2}\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right)}$$

$$= \zeta \frac{1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}}{1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}} \cdot \frac{\frac{q^2-1}{2}\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right) - (q-c)i\sin\frac{4\pi}{\lambda}}{\frac{q^2-1}{2}\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right) + (q-c)i\sin\frac{4\pi}{\lambda}}$$

$$u(h(q(z))) = \zeta \frac{1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}}{1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}} \cdot \frac{\frac{\mathrm{e}^{\frac{4i\pi}{\lambda}z}-1}{2}\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right) - (\mathrm{e}^{\frac{2i\pi}{\lambda}z}-c)i\sin\frac{4\pi}{\lambda}}{\frac{\mathrm{e}^{\frac{4i\pi}{\lambda}z}-1}{2}\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right) + (\mathrm{e}^{\frac{2i\pi}{\lambda}z}-c)i\sin\frac{4\pi}{\lambda}}$$

$$= \frac{\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right) i\sin\frac{2\pi}{\lambda}z - (1-c\mathrm{e}^{-\frac{2i\pi}{\lambda}z})i\sin\frac{4\pi}{\lambda}}{\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right) i\sin\frac{2\pi}{\lambda}z + (1-c\mathrm{e}^{-\frac{2i\pi}{\lambda}z})i\sin\frac{4\pi}{\lambda}}$$

$$= \frac{\left(1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}\right) \sin\frac{2\pi}{\lambda}z - (1-c\mathrm{e}^{-\frac{2i\pi}{\lambda}z})\sin\frac{4\pi}{\lambda}}{\left(1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}\right) \sin\frac{2\pi}{\lambda}z + (1-c\mathrm{e}^{-\frac{2i\pi}{\lambda}z})\sin\frac{4\pi}{\lambda}}$$

$$= \frac{(1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z - (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}}{(1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z + (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}}$$

where we have made the obvious choice $\zeta = \frac{1-c\mathrm{e}^{-\frac{4i\pi}{\lambda}}}{1-c\mathrm{e}^{\frac{4i\pi}{\lambda}}}$.

The mapping is not 1 to 1: it is 2 to 1 on the vertical strip $0 \leqslant \operatorname{Im} z \leqslant \frac{\lambda}{2}$, but as a consequence of Lemma 9 below, it is 1 to 1 on our domain.

Graphical note: observe that $h(m) = m$ and $h(M) = M$. The domain Ω'_q covers better the upper half-plane of $\mathbf{C}_q$. It is somewhat similar to Ω_q but the curve corresponding to the border of Ω_z is smooth at points corresponding to $z = \pm i$.

Lemma 9 Ω_q *does not intersect the circle of diameter* $[m, M]$.

Proof: The intersections of Ω_q and the circle of diameter $[m, M]$ are a subset of the solutions of the equation

$$|\mathrm{e}^{2\pi iz/\lambda} - \cosh(2\pi/\lambda)|^2 = \sinh^2(2\pi/\lambda).$$

Such equation can be written also as

$$\mathrm{e}^{\epsilon(w+\bar{w})} - (\mathrm{e}^{\epsilon w} + \mathrm{e}^{\epsilon\bar{w}})\cosh(\epsilon) + 1 = 0$$

where $\epsilon := 2\pi/\lambda$ and $w := iz$. All the functions appearing in previous equation are entire in the variable ϵ, hence using the known representations as Taylor series we obtain that w is a solution if and only if it satisfies

$$\frac{(w+\bar{w})^n}{n!} - \sum_{\substack{m,k\geq 0\\ m+2k=n}} \frac{w^m+\bar{w}^m}{m!}\cdot\frac{1}{(2k)!} + \delta_{n,0} = 0 \qquad \forall n \geq 0.$$

Cases $n=0$ and $n=1$ are always satisfied. For $n=2$ we have

$$\frac{(w+\bar{w})^2}{2} = \frac{w^2+\bar{w}^2}{2} + \frac{2}{0!}\cdot\frac{1}{2}$$

so that $w\bar{w}=1$; for $n=4$ we have

$$\frac{(w+\bar{w})^4}{4!} = \frac{w^4+\bar{w}^4}{4!} + \frac{w^2+\bar{w}^2}{2!}\cdot\frac{1}{2!} + \frac{2}{0!}\cdot\frac{1}{4!}$$

i.e., $w^2+\bar{w}^2=2$. Since $\bar{w}=w^{-1}$, this equation implies that $w=\pm 1$, so that the unique solutions of the original equation are $z=\pm i$ which do not belong to Ω_q. $\square$

3.4 Improving the square root

In the definition of f_α, we can substitute w^{-1} by $w_\ell(u)=u^\ell$, where $\ell \in (0;1)$. To do this, it is sufficient to choose a branch of the logarithm. We'll precise it later. The resulting formula is

$$f_{\ell,\alpha}(u) = \zeta_{\ell,\alpha}\cdot\frac{\left(-\alpha\cdot\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell - (-\alpha)^\ell}{1-\overline{(-\alpha)^\ell}\left(-\alpha\cdot\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell},$$

where $\zeta_{\ell,\alpha}$ is a complex sign to be precised later. To choose a branch of the logarithm, we can choose a logarithm for $-\alpha$ and take the natural choice of the logarithm for the image of Ω under the map $r : u \longmapsto \frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}$. Indeed, r maps Ω to a domain containing 1 with 0 as a border point. We choose an arbitrary logarithm for $-\alpha$ and use the branch of the logarithm that takes value 0 in 1. We then have:

$$f_{\ell,\alpha}(u) = \zeta_{\ell,\alpha}(-\alpha)^\ell\cdot\frac{\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell - 1}{1-|\alpha|^{2\ell}\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell}.$$

It is not difficult to check that $f_{\ell,\alpha}(u)$ is asymptotic to $\zeta_{\ell,\alpha}(-\alpha)^\ell\ell\frac{\overline{\alpha}-\frac{1}{\alpha}}{1-|\alpha|^{2\ell}}u = -\zeta_{\ell,\alpha}\frac{(-\alpha)^\ell}{\alpha}\ell\frac{1-|\alpha|^2}{1-|\alpha|^{2\ell}}u$ as $u\to 0$ and thus to get a positive derivitaive in 0, we choose $\zeta_{\ell,\alpha} = -s\frac{|\alpha|^\ell}{(-\alpha)^\ell}$ (where, as before, $s=\frac{\alpha}{|\alpha|}$). We get

$$f_{\ell,\alpha}(u) = s|\alpha|^\ell\cdot\frac{1-\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell}{1-|\alpha|^{2\ell}\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)^\ell}.$$

We then have $f_{\ell,\alpha}(0) = 0$, $f_{\ell,\alpha}(\alpha) = s|\alpha|^\ell$ and

$$f'_{\ell,\alpha}(0) = \ell|\alpha|^{\ell-1}\frac{1-|\alpha|^2}{1-|\alpha|^{2\ell}} = \sinh(\ln|\alpha|)\cdot\frac{\ell}{\sinh(\ell\ln|\alpha|)} = \frac{\sinh(\ln|\alpha|)}{\ln|\alpha|}\cdot\frac{\ell\ln|\alpha|}{\sinh(\ell\ln|\alpha|)}.$$

The function $\frac{\sinh t}{t}$ is even and has Taylor series with positive coefficients and infinite radius of convergence. It follows that it is decreasing in $(-\infty;0)$ and increasing in $(0;+\infty)$. This in turns means that $f'_{\ell,\alpha}(0)$ is bigger as ℓ gets smaller. It is obvious that $f_{\ell,\alpha}(\alpha)$ also gets bigger as ℓ gets smaller. It is natural to consider the limit of $f_{\ell,\alpha}$ as $\ell \to 0$, i.e.

$$\begin{aligned} f_{0,\alpha}(u) &= s\frac{\ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)}{\ln|\alpha|^2+\ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)} = s\left(1-\frac{\ln|\alpha|^2}{\ln|\alpha|^2+\ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)}\right) \\ &= s\left(1-\frac{2\ln|\alpha|}{\ln\left(\frac{|\alpha|^2-\overline{\alpha}u}{1-\overline{\alpha}u}\right)}\right) = s\left(1-\frac{2\ln|\alpha|}{\ln\left(1-\frac{1-|\alpha|^2}{1-\overline{\alpha}u}\right)}\right) \end{aligned}$$

Notice that for any $\ell > 0$, $f_{\ell,\alpha}$ maps Ω into U. Indeed if $\mathcal{O} \subset U$ is any connected, simply connect open non containing 0, then each w_ℓ maps $\mathcal{O}$ into U. This means that $f_{0,\alpha}$ maps Ω into $\overline{U}$. Since $f_{0,\alpha}$ is holomorphic, it is open and thus maps Ω into U. From the last expression, it is obvious that $f_{0,\alpha}$ is injective (u appears only once in the expression).

We have

$$f'_{0,\alpha}(0) = s\frac{\overline{\alpha}-\frac{1}{\alpha}}{\ln|\alpha|^2} = \frac{|\alpha|^2-1}{2|\alpha|\ln|\alpha|} = \frac{\sinh\ln|\alpha|}{\ln|\alpha|} > 1.$$

When $u = \alpha(1+it)$, we have

$$\frac{1}{s}f_{0,\alpha}(u) = 1 - 2\frac{\ln|\alpha|}{\ln|t|} - \frac{\ln|\alpha|\ln(2-\frac{2}{|\alpha|^2}+\frac{1}{|\alpha|^4}) + i\ln|\alpha|(\pi+2\arctan\frac{1-|\alpha|^2}{|\alpha|^2}-\arg t)}{(\ln|t|)^2} + o((\ln|t|)^{-2}),$$

which means that $f_{0,\alpha}(\Omega)$ will touch ∂U with a cusp whose two semitangents are directed towards 0.

3.5 Back to the half plane !

As functions of α and u, observe that both $\frac{1}{s}f_\alpha(u)$ and $\frac{1}{s}f_\alpha(u)$ depend only on $|\alpha|$ and $\frac{u}{s}$ (where $s = \frac{\alpha}{|\alpha|}$). If we define f_n as in Theorem 6 and g_n again as in Theorem 6 but substituting f_{α_n} by $\frac{|\alpha_n|}{\alpha_n}f_{\alpha_n}$, then $f_n = \zeta_n g_n$ where $\zeta_n \in S^1$ is the product of the complex signs of all the α_k, $1 \leqslant k \leqslant n$. In particular, $\zeta_n = \frac{g'_n(0)}{|g'_n(0)|}$. The sequence $\{g_n\}$ will therefore not converge, but $\{\zeta_n^{-1}g_n\}$ does converge uniformly over compacts towards f. There is a slight algorithmic advantage of g_n over f_n because we remove a complex product at each iteration.

For $(\alpha, u) \in U^2$, define

$$g_{0,\alpha}(u) = \frac{\ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)}{\ln|\alpha|^2 + \ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)}.$$

Define $g_{0,n}$ as in Theorem 6, subsituting f_{α_n} by g_{0,α_n}. Then as above, if $\zeta_{0,n}$ is the complex sign of $g'_{0,n}(0)$, the sequence $\{\zeta_{0,n}^{-1} g_{0,n}\}$ will converge towards f, uniformly over compacts.

Now we make another change of variable taking

$$r = \frac{1+u}{1-u}$$
$$u = \frac{r-1}{r+1}.$$

We then have

$$r(g_{0,\alpha}(u(r))) = \frac{\ln|\alpha|^2 + 2\ln\left(\frac{1-\frac{u}{\alpha}}{1-\overline{\alpha}u}\right)}{-\ln|\alpha|^2}$$

and, with $\gamma = r(\alpha)$,

$$\begin{aligned} r(g_{0,\alpha}(u(r))) &= 1 - 2\frac{\ln\left(\frac{1-\frac{(\gamma+1)(r-1)}{(\gamma-1)(r+1)}}{1-\frac{(\overline{\gamma}-1)(r-1)}{(\overline{\gamma}+1)(r+1)}}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|^2} \\ &= 1 - 2\frac{\ln\left(\frac{\overline{\gamma}+1}{\gamma-1}\cdot\frac{(\gamma-1)(r+1)-(\gamma+1)(r-1)}{(\overline{\gamma}+1)(r+1)-(\overline{\gamma}-1)(r-1)}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|^2} = 1 - 2\frac{\ln\left(\frac{\overline{\gamma}+1}{\gamma-1}\cdot\frac{\gamma-r}{\overline{\gamma}+r}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|^2} \end{aligned}$$

We will denote this last function $g_{0,\gamma}$. With respect to $g_{0,\alpha}$, we get an improvement because the non constant logarithm is now at the numerator and not at the denominator.

Going real ? We have

$$\begin{aligned} r(z) &= \frac{\left((1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z + (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}\right) + \left((1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z - (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}\right)}{\left((1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z + (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}\right) - \left((1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z - (1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}\right)} \\ &= \frac{(1-c\cos\frac{4\pi}{\lambda})\sin\frac{2\pi}{\lambda}z}{(1-c\cos\frac{2\pi}{\lambda}z)\sin\frac{4\pi}{\lambda}}. \end{aligned}$$

Since Ω_z is stable by complex conjugation, $\Omega_r = r(\Omega_z)$ will have the same property. We would like to preserve that property at each step. The idea is to use the mean of $g_{0,\gamma}$ and

$g_{0,\overline{\gamma}}$. Indeed, the denominator of $g_{0,\gamma}$ does not change if we substitute γ by $\overline{\gamma}$ and we get:

$$\begin{aligned} G_{0,\gamma}(r) = \frac{g_{0,\gamma}(r) + g_{0,\overline{\gamma}}(r)}{2} &= 1 - \frac{\ln\left(\frac{\overline{\gamma}+1}{\gamma-1}\cdot\frac{\gamma-r}{\overline{\gamma}+r}\right) + \ln\left(\frac{\gamma+1}{\overline{\gamma}-1}\cdot\frac{\overline{\gamma}-r}{\gamma+r}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|} \\ &= 1 - \frac{\ln\left(\frac{(\overline{\gamma}+1)(\gamma+1)}{(\gamma-1)(\overline{\gamma}-1)}\cdot\frac{\gamma-r}{\overline{\gamma}+r}\cdot\frac{\overline{\gamma}-r}{\gamma+r}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|} \\ &= 3 - 2\frac{\ln\left(\frac{\gamma-r}{\gamma+r}\cdot\frac{\overline{\gamma}-r}{\overline{\gamma}+r}\right)}{\ln\left|\frac{\gamma-1}{\gamma+1}\right|^2} \end{aligned}$$

The applcation $G_{0,\gamma}$ is real in the sense that $\overline{G_{0,\gamma}(\overline{r})} = G_{0,\gamma}(r)$ (equivalently, all the Taylor coefficients of its development at any real point are real). This means that the image of a domain that is stable for complex conjugation will be stable by complex conjugation. Unfortunately, it is not very difficult to see that $G_{0,\gamma}\left(\frac{|\gamma|^2}{r}\right) = G_{0,\gamma}(r)$.

4 Poles and convergence radius

We have constructed a function f from $\mathbf{C} = \mathbf{C}_z$ (minus some real points) to $\mathbf{P}^1(\mathbf{C})$. We call its Fourier expansion its q-expansion. We compute where the poles of f lie. This gives a convergence radius for its q-expansion.

Let $\Lambda_0(z)$ denote the $\mathsf{G}(\lambda)$-orbit of a point z and let $\Lambda(z)$ be the set of limits points of $\Lambda_0(z)$. The discreteness of the action of $\mathsf{G}(\lambda)$ on $\mathfrak{h}$ implies that $\Lambda(z) \subseteq \mathbf{P}^1(\mathbf{R})$. The presence of the parabolic transformation T in $\mathsf{G}(\lambda)$ assures that $\Lambda_0(z)$ is not finite for any z thus proving that the Fuchsian group $\mathsf{G}(\lambda)$ is non-elementary (see [Bea, Def. 5.1.12]). It is known that for such groups $\Lambda(z)$ does not depend on z (see [Bea, Th. 5.3.9]) so that henceforth we denote it simply by Λ. The limit set Λ is a perfect set (see [Bea, Th. 5.3.7]). To obtain a more explicit description of Λ is a difficult task, but certainly $\Lambda \cap (1, \lambda-1) = \emptyset$, since $(1, \lambda-1)$ belongs to $\mathcal{B}^* \cup T\Sigma_2(\mathcal{B}^*)$ which is another fundamental domain for $\mathsf{G}(\lambda)$.

The unique pole of f in the u variable is at ∞, so that in the z variable it will lie in the orbit under the action of $\mathsf{G}(\lambda)$ of one inverse image of ∞. Observe that $u = \infty$ for $q = \overline{t}$ so the poles of f will lie in the orbit of $z_0 = -\frac{\lambda}{4} - i\frac{\lambda \ln t}{2\pi}$. As observed above, the set of limit points of this orbit, Λ, do not depend on z_0 (and thus t), are real and lie outside $(1, \lambda-1)$.

Conclusions

We substantially improved the techniques of Köbe. However we tried to compute a substantial number of coefficients of Hecke's modular forms but where unable to obtain plausible results concerning their behavior.

References

[Ahl] L. V. Ahlfors. *Complex analysis.* McGraw-Hill Book Co., New York, third edition, 1978. 1.2

[Bea] A. F. Beardon. *The geometry of discrete groups*, volume 91 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1983. 4

[Ber] B. C. Berndt. *Hecke's theory of modular forms and Dirichlet series.* Springer Verlag, 1970. 1.1, 1.2

[Del] P. Deligne. La conjecture di Weil. I. *Publ. Math. Inst. Hautes Études Sci.*, (43):273–307, 1974. 1.1

[DS] P. Deligne and J.-P. Serre. Formes modulaires de poids 1. *Ann. Sci. École Norm. Sup. (4)*, **7**:507–530 (1975), 1974. 1.1

[Hec] Erich Hecke. *Lectures on Dirichlet series, modular functions and quadratic forms.* Vandenhoeck & Ruprecht, Göttingen, 1983. Edited by Bruno Schoeneberg, With the collaboration of Wilhelm Maak. 1, 1.1

[KMP$^+$] Jerzy Kaczorowski, Giuseppe Molteni, Alberto Perelli, Jörn Steuding, and Jürgen Wolfart. Hecke theory and the Selberg class. *Funct. Approx. Comment. Math.*, **35**:183–193, 2006. 1.1

[Köb] P. Köbe. Abhandlungen zur Theorie der konformen Abbildung. I. Die Kreisabbildung des allgemeinsten einfach und zweifach zusammenhängenden schlichten Bereichs und die Ränderzuordnung bei konformer Abbildung. *Journal für die reine und angewandte Mathematik*, **145**:177–223, 1915. 1.2

[Rud] Walter Rudin. *Real and complex analysis.* McGraw-Hill Book Co., New York, third edition, 1987. 1.2, b

[SS] E. M. Stein and R. Shakarchi. *Complex analysis.* Princeton Lectures in Analysis, II. Princeton University Press, Princeton, NJ, 2003. 2.3, e, f

First version of the program

```
/*-*- compile-command: "
/usr/bin/gcc -c -o reduced.gp.o -fPIC -O4 -march=athlon64 -Wall \
    -fno-strict-aliasing -fomit-frame-pointer \
    -I/home/grenie/parinst/include reduced.gp.c && \
    /usr/bin/gcc -o reduced.gp.so -shared -fPIC -O4 \
    -march=athlon64 -Wall -fno-strict-aliasing \
    -fomit-frame-pointer -Wl,-shared reduced.gp.o -lc -ldl -lm \
    -L/home/grenie/parinst/lib -lpari
"; -*-*/
/*
gcc -I ~/parinst/include -O3 -o reduced reduced.c\
    -L ~/parinst/lib -lpari -lrt
 */
/*
 * Sul cluster di Dalmine:
mpicc -I$HOME/install/include -g -DMPI -O4 -Wall reduced.c \
    -o reduced ~/install/lib/libpari.a -lrt -lm
 */

#ifdef __CYGWIN__
typedef unsigned long ulong;
#endif
#include <pari/pari.h>
#include <pari/paripriv.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <net/if.h>
#include <netinet/in.h>
#include <errno.h>
#include <pthread.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <ctype.h>
#include <sys/resource.h>
#include <unistd.h>
```

```
#ifdef MPI
#include <mpi.h>
#else
#include <netdb.h>
#endif

 static void *hdlr[NSIG];

#define RED_PORT          2567
 /* below: (srand48(time(NULL)+getpid());lrand48()+lrand48()) */
#define RED_MAGIC         (htonl(0x77d49029))
#define RED_MAGIC2        (htonl(0x2990d477))
#ifndef WITH_FORK
#define WITH_FORK 0
#endif

 static int parallel = 1, np, level = 0;
#ifndef MPI
 static GEN *genpeers;
 static int state;
 static int *peerlevel;
 static struct peer *peers;
 static int *sockpeers;
 static int sockmaster, imaster = -1;
 static FILE **filepeers;
 static FILE *filemaster;
 static struct sockaddr_in *otherhosts;
 static int numother;
#else
 static void *complex_buf;
#endif
 static GEN doms, f;      /* Per wg */
 static GEN uno;
 static GEN cur_dom;
 static ulong iter, maxiter;
 static int destructive = 0;
 static long *lgthes = 0, curidx = 0;
 static long idxincr, globaln, c0, chunk0, last_peer;
 static int pending_sig;

 /*static unsigned long long series = 0, points = 0; */

 static void lbreak(void) {}
```

```
/*
GP;install("sgn","D0,G,p","sgn","./reduced.gp.so");
GP;install("minc","lG","minc","./reduced.gp.so");
GP;install("maxc","lG","maxc","./reduced.gp.so");
GP;install("domain","GDGDGp","domain","./reduced.gp.so");
GP;install("varphi","GG","varphi","./reduced.gp.so");
GP;install("stepkoebe","GDGp","stepkoebe","./reduced.gp.so");
GP;install("writemat","vsG","writemat","./reduced.gp.so");
GP;install("taylorcoeffs","G","taylorcoeffs","./reduced.gp.so");
GP;install("writegnuplot","vsG","writegnuplot","./reduced.gp.so");
GP;install("wg","vIp","wg","./reduced.gp.so");
GP;install("init_reduced","v","init_reduced","./reduced.gp.so");
*/
GEN sgn(GEN z, long prec);
long minc(GEN L);
long maxc(GEN L);
GEN domain(GEN lambda, GEN t, GEN n, long prec);
GEN varphi(GEN alpha, GEN z);
GEN stepkoebe(GEN alpha, GEN dom, long prec);
void writemat(char *file, GEN var);
GEN taylorcoeffs(GEN dom);
void writegnuplot(char *file, GEN dom);
void wg(long k, long prec);
void init_reduced(void);
/*End of prototype*/

struct peer {
  uint32_t              p_magic;
  struct sockaddr_in    p_addr;
  in_addr_t             p_addr_min;
  unsigned              p_contacted;
};

void
init_reduced(void)          /* void */
{
  return;
}

void
usr1handler(void)
{
  char *s;
  time_t t;

  time(&t);
  s = ctime(&t);
  if (s[strlen(s)-1] == '\n')
      s[strlen(s)-1] = '\0';
  printf("%s i=%lu\n", s, iter);
  fflush(stdout);
}

void
save_state(int sig)
{
  if (cur_dom && np <= 1) {
    writebin("emergency",
             mkvec3(stoi(iter), stoi(maxiter), cur_dom));
    printf("Done (%lu).\n", iter);
    pending_sig = 0;
  }
  else {
    pending_sig = sig;
    if (!cur_dom)
      printf("Sorry: garbage collecting (%lu).\n", iter);
  }
  fflush(stdout);
  if (np <= 1 && sig != SIGUSR2) {
    sigset_t set;

#ifdef MPI
    MPI_Barrier(MPI_COMM_WORLD);
#endif
    signal(sig, hdlr[sig]);
    sigemptyset(&set);
    sigaddset(&set, sig);
    sigprocmask(SIG_UNBLOCK, &set, NULL);
#ifdef MPI
    MPI_Barrier(MPI_COMM_WORLD);
#endif
    kill(getpid(), sig);
    /* Segnale non fatale ! */
    signal(sig, (void *)&save_state);
  }
  else
    signal(sig, (void*)&save_state);
}

GEN
sgn(GEN z, long prec)
{
  if (!z || z == gen_0)
      return gen_0;
  return gdiv(z, gabs(z, prec));
}

long
minc(GEN L)
{
  GEN loicmin, loictmp;
  long iloicmin;
  long l1;

  loicmin = gnorm(gel(L, 1));
  iloicmin = 1;
  l1 = glength(L);
  {
    int i;
    for (i = 2; i <= l1; i++)
    {
      loictmp = gnorm(gel(L, i));
      if (gcmp(loictmp, loicmin) < 0)
      {
        iloicmin = i;
        loicmin = loictmp;
      }
    }
  }
  return iloicmin;
}

long
maxc(GEN L)
{
  GEN loicmax, loictmp;
  long iloicmax;
  long l1;
  loicmax = gnorm(gel(L, 1));
  iloicmax = 1;
  l1 = glength(L);
  {
    int i;
    for (i = 2; i <= l1; i++)
    {
      loictmp = gnorm(gel(L, i));
      if (gcmp(loictmp, loicmax) > 0)
      {
        iloicmax = i;
        loicmax = loictmp;
      }
    }
  }
  return iloicmax;
}

GEN
domain(GEN lambda, GEN t, GEN n, long prec)          /* vec */
{
    GEN v, name, namec;
    GEN z = pol_x(fetch_user_var("z"));
    GEN q = pol_x(fetch_user_var("q"));
    GEN u = pol_x(fetch_user_var("u"));
    GEN dom;        /* vec */
#if 0
    GEN z0, q0, zp, qp, up, u0;
    GEN tre, z1, z2;
#endif
    GEN tmp, tmp2, c, vecf;
    int i;
    long nl;

    if (!n)
      n = stoi(1000);
    dom = cgetg(4, t_VEC);
    vecf = cgetg(4, t_VEC);
    nl = itos(n);
    {
      ulong first, i, ldom;
      GEN r;
```

```
c0 = glength(vecf) * precdl * precdl / 6;
ldom = (nl + 1 + c0) / np;
c0 = nl + 1 - (np - 1) * ldom;
if (c0 <= 0) {
  ldom -= (-c0) / (np - 1) + 1;
  c0 = nl + 1 - (np - 1) * ldom;
}
chunk0 = c0 * np;
if (!level) ldom = c0;
if (np == 1)
  last_peer = 0;
else
  last_peer = 1 + (nl + 1 - chunk0) % (np - 1);
if (c0 > 1)
  tmp = gdiv(gmulsg(-2 * np, uno), n);
v = cgetg(ldom+1, t_VEC);
/* Special cases */
if (!level)
  gel(v, 1) = r = uno;
if (level == last_peer)
  gel(v, ldom--) = gneg(uno);
for (i = 1+!level; i <= ldom; ++i) {
  GEN pt, im;

  /* The k-th point is e^{i\arcsin{\frac{2k-n}{n}}} */
  gel(v, i) = pt = cgetg(3, t_COMPLEX);
  /* We already multiply by i (recall that tmp=-2/n) */
  if (i == 1)
    r = gdiv(gmulsg(level - nl, uno), n);
  else
    r = gadd(r, tmp);
  gel(pt, 1) = r;
  im = gsub(uno, gsqr(r));
  if (signe(im) < 0) /* im[0] & HIGHBIT */
    gel(pt, 2) = gen_0;
  else
    gel(pt, 2) = gsqrt(im, prec);
  if (i == c0)
    tmp = gdiv(gmulsg(-2 * (np - 1), uno), n);
}
/* Correct ldom for the last node
 * (modified in special cases above) */
```

```
    if (level == last_peer) ldom++;
    /* Acts as an offset for the computations of the indices */
    c0++;
    /* We do not need to multiply by i here,
     * it has been done before */
    tmp = gdiv(gmul(gen_2, mppi(prec)), lambda);
    for (i = 1; i <= ldom; ++i)
      gel(v, i) = gexp(gmul(gel(v, i), tmp), prec);
    c = gch(tmp, prec);
    for (i = 1; i <= ldom; ++i) {
      tmp = gel(v, i);
      gel(v, i) = gdiv(gsub(gsqr(tmp), uno),
                       gmulsg(2, gsub(tmp, c)));
    }
    if (t) {
      tmp2 = mulcxI(t);
      tmp = gneg(tmp2);
    }
    else {
      /* Instead of ti we take the image of 2 */
      tmp = gexp(gdiv(gmulsg(4, mulcxI(mppi(prec))),
                 lambda), prec);
      tmp = gneg(gdiv(gsub(gsqr(tmp), uno),
                 gmulsg(2, gsub(tmp, c))));
      tmp2 = gconj(tmp);
    }
    for (i = 1; i <= ldom; ++i) {
      GEN pt = gel(v, i);
      gel(v, i) = gdiv(gadd(pt, tmp), gadd(pt, tmp2));
    }
  }
  gel(dom, 1) = v;
  if (!level) {
    i = 1;
    gel(vecf, i++) = gcopy(u);
    q = gdiv(gsub(gsqr(q), uno), gmulsg(2, gsub(q, c)));
    /* Please look in stepkoebe why
     * those two series are enough */
    gel(vecf, i++) = gdiv(gadd(q, tmp), gadd(q, tmp2));
    q = gexp(gmul(gdiv(gmul(mkcomplex(gen_0,gen_2),mppi(prec)),
                       lambda),
                  gadd(z,
                       mulcxI(strtor("1.1", prec)))), prec);
    q = gdiv(gsub(gsqr(q), uno), gmulsg(2, gsub(q, c)));
    gel(vecf, i++) = gdiv(gadd(q, tmp), gadd(q, tmp2));
    if (i != lg(vecf))
      pari_err(talker,
          "Improper␣vecf␣length␣in␣domain␣(%d␣instead␣of␣%d)",
          lg(vecf), i);
    gel(dom, 2) = vecf;
  }
  else
    gel(dom, 2) = gen_0;
  if (!t)
    namec = strtoGENstr("-0");
  else
  {
    namec = concat(strtoGENstr("-"), GENtoGENstr(t));
    if (typ(t) == t_REAL)
    {
      char *namep = GSTR(namec);
      long i;

      for (i = glength(namec); i > 0; i--)
        if (namep[i-1] != '0' && i <= 21)
        {
          namep[i] = '\0';
          namec = strtoGENstr(namep);
          break;
        }
    }
  }
  name = GENtoGENstr(lambda);
  if (typ(lambda) == t_REAL)
  {
    char *namep = GSTR(name);
    long i;

    for (i = glength(name); i > 0; i--)
      if (namep[i-1] != '0' && i <= 20)
      {
        if (namep[i-1] == '.') {
          if (i == 1)
            namep = "0"; /* unlikely */
          else
            namep[i-1] = '\0';
          }
          else
            namep[i] = '\0';
          name = strtoGENstr(namep);
          break;
        }
    }
    name = concat(name, namec);
#if 0
    vecf = cgetg(7, t_VEC);
    gel(vecf, 1) = lambda;
    gel(vecf, 2) = t;
    gel(vecf, 3) = name;
    gel(vecf, 4) = up;
    gel(vecf, 5) = u0;
    gel(vecf, 6) = n;
#else
    vecf = cgetg(5, t_VEC);
    i = 1;
    gel(vecf, i++) = lambda;
    gel(vecf, i++) = t ? t : mulcxI(gneg(tmp));
    gel(vecf, i++) = name;
    gel(vecf, i++) = n;
    if (i != lg(vecf))
        pari_err(talker,
        "Improper third component length (%d insted of %d).\n",
        lg(vecf), i);
#endif
    /* Create third element (never copied) on the heap */
    gel(dom, 3) = gclone(vecf);
    return dom;
}

GEN
varphi(GEN alpha, GEN z)
{
  return gdiv(gsub(z, alpha), gsubsg(1, gmul(gconj(alpha), z)));
}
```

```
/*
 * Nota sul segno della radice quadrata.
 * ---
 * Commento di "conto":
 *                                                              -----------------------------
 *                  (            1    )-1     z         /(    |\alpha|z)(
z      )
 *Verifica che s(|\beta|+-------)  (1+ - - \   / (1- ---------)(1- ---------
 *                  (          |\beta|)       s      \/  (         s      )(
s|\alpha|)
 * (dove s = sgn(\alpha)=\alpha/|\alpha|).
 * è uno "step" di Koebe. Ricordiamo che uno step di Koebe è
 * sgn(beta) volte una reciproca di
 * \phi_{-beta}\circ\sqrt{\,}\circ\phi_{-\alpha} (con \beta
 * una radice quadrata di -\alpha e \phi definita sopra).
 *
 * Inoltre, l'argomento della radice quadrata è
 * 1+(|\alpha|+1/|\alpha|+z/s)z/s=
 * (1-(|\alpha|+1/|\alpha|)z/s+z/s^2=
 * (1-conj(\alpha)z)(1-z/\alpha).
 * ---
 * (1-conj(\alpha)z)(1-z/\alpha) ha parte immaginaria di grado
 * due in (x,y), quindi è reale su una conica. Questa conica
 * contiene la retta z=t\alpha ed è quindi degenere. La seconda
 * retta è * z=(1+|\alpha|^2)\alpha/(2|\alpha|^2)+it\alpha, è
 * quindi perpendicolare alla prima e esterna alla circonferenza
 * unitaria (perché la proiezione dell'orginie è il punto
 * (1+|\alpha|^2)\alpha/(2|\alpha|^2) che ha modulo
 * (1+|\alpha|^2)/(2|\alpha|)>1. Sulla prima retta, il reale è
 * negativo per 1<t<1/|\alpha|^2) (cioè per z su
 * [\alpha;1/conj(\alpha)]). Sulla seconda il reale è sempre neg.
 * Dobbiamo cambiare il segno della radice quadrata quando
 * un percorso incontra quindi il suddetto segmento o la suddetta
 * retta. Ma il nostro problema è limitato alla circonferenza
 * unitaria, quindi questo non può quasi mai succedere (anche
 * perché il dominio è semplicemente connesso). Per le funzioni,
 * è necessario scegliere il segno giusto solo se non funziona
 * per i punti del bordo. A questo punto il conto va a ramengo...
 * ---
 * Osservazione: si può calcolare il segno "giusto" della radice
 * quadrata, usando (per i punti del bordo) la continuità e (per
 * le serie) varie serie e una valutazione su punti comuni.
 */

GEN
stepkoebe(GEN alpha, GEN dom, long prec)            /* vec */
{
  GEN aalpha, s, zz, f=0, f1=0, divs, alpharen;
  GEN coeff, coeff2, coeff3;
  long lfuncs;
  long modl;
  int imin = -1;
  GEN rdom;          /* vec */
  GEN dom1, dom2, dom3, dom34;
#if WITH_FORK
  long mid;
  GEN *zzp, zz2;
  char name[50];
  int fd;
  void *mem;
#endif

  if (!dom) {
    dom = alpha;
    alpha = NULL;
  }
  if (!dom)
    pari_err(talker, "stepkoebe: needs dom !\n");
#ifdef MPI
  {
      int tmp_sig = -1, mask = sigblock(-1);

      /* In at least one implementation, MPI_Allreduce has no
       * effect if tmp_sig==pending_sig */
      MPI_Allreduce(&pending_sig, &tmp_sig, 1, MPI_INT,
              MPI_MAX, MPI_COMM_WORLD);
      pending_sig = tmp_sig;
      sigsetmask(mask);
  }
#else
  if (np > 1 && !level) {
    int i;
    char c;
    for(i = 1; i < np; i++) {
      fread(&c, 1, 1, filepeers[i]);
      if (c > pending_sig)
        pending_sig = c;
    }
    c = pending_sig;
    for(i = 1; i < np; i++)
      fwrite(&c, 1, 1, filepeers[i]);
  }
  else if (np > 1) {
    char c;

    c = pending_sig;
    fwrite(&c, 1, 1, filemaster);
    fread(&c, 1, 1, filemaster);
    pending_sig = c;
  }
#endif
  if (pending_sig) {
    int old_np = np;

    np = 1;
    save_state(pending_sig);
    np = old_np;
  }
  dom1 = gel(dom, 1);
  modl = glength(dom1);
  if (!alpha) {
    int peer_idx;
#ifdef MPI
    void *ptr;
#endif

    if (lgthes) {
      curidx += idxincr;
      if (curidx >= globaln)
        curidx -= globaln;
      if (curidx < chunk0)
        peer_idx = curidx % np;
      else
        peer_idx = 1 + ((curidx - chunk0) % (np - 1));
      if (level == peer_idx) {
        if (curidx < chunk0)
```

```
          imin = 1 + curidx / np;
        else
          imin = c0 + (curidx - chunk0) / (np - 1);
        alpha = gel(dom1, imin);
#ifdef MPI
        if (gel(alpha, 1) != gel(alpha, 2) + lg(gel(alpha, 2))) {
          ptr = complex_buf;
          memcpy(complex_buf, gel(alpha, 2),
                 lg(gel(alpha, 2)) * sizeof(ulong));
          memcpy(complex_buf + lg(gel(alpha, 2)) * sizeof(ulong),
                 gel(alpha, 1),
                 lg(gel(alpha, 1)) * sizeof(ulong));
        }
        else
          ptr = (void *)gel(alpha, 2);
        MPI_Bcast(ptr, lg(gel(alpha, 1)) + lg(gel(alpha, 2)),
            MPI_LONG, peer_idx, MPI_COMM_WORLD);
#else
        syntax error
#endif
      }
      else {
#ifdef MPI
        GEN real;
        int n;
#endif

        imin = 0;
#ifdef MPI
        alpha = cgetg(3, t_COMPLEX);
        gel(alpha, 2) = real = cgetg(2 * prec + 2, t_REAL);
        MPI_Bcast(real, 2 * prec + 2, MPI_LONG, peer_idx,
                  MPI_COMM_WORLD);
        gel(alpha, 1) = real + lg(real);
        n = lg(real) + lg(gel(alpha, 1));
        if (n < 2 * prec) {
          int i;

          /* Clean up stack, should probably use stackdummy */
          real += n;
          settyp(real, t_VEC);
          setlg(real, 2*prec-n);
          for (i = 1; i < 2*prec - n; i++)
```

```
            gel(real, i) = gen_0;
        }
#else
        syntax error
#endif
      }
      aalpha = gabs(alpha, prec);
    }
    else {
      int i;
      GEN loictmp;

/*printf("bl%d %08x %08x %08x %08x\n", level, ((unsigned int *)alpha)[0],
      aalpha = gen_2;
      imin = 0;
      for (i = modl - (level == last_peer); i > 0; i--) {
        GEN el;

        if (!level && i == 1)
            break;
        if ((el = gel(dom1, i)) == gen_0)
            continue;
        loictmp = gnorm(el);
        if (gcmp(loictmp, aalpha) < 0) {
          aalpha = loictmp;
          alpha = el;
          imin = i;
        }
      }
/*printf("%d,", imin); */
/*print(mkvec(mkvec2(stoi(imin), gsqrt(aalpha, prec)))); */
/*if (imin == 460) print(mkvec(gabs(gel(dom1, 462), prec))); */
#ifdef MPI
      {
        struct {
          double absmen1;
          int lev;
        } in, out;

        if (aalpha == gen_2)
          in.absmen1 = 1;
        else
          in.absmen1 = rtodbl(subrr(aalpha, uno));
        in.lev = level;
        /* Get min abs(dom[1])-1 */
        MPI_Reduce(&in, &out, 1, MPI_DOUBLE_INT, MPI_MINLOC, 0,
                     MPI_COMM_WORLD);
        /* level 0 broadcast the level of the minimum to all
         * other levels */
        if (out.absmen1 > 0)
            /* Finished: only gen_0 in the points */
            out.lev = -1;
        MPI_Bcast(&out.lev, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (out.lev == -1)       /* Finished */
            return cur_dom;
        if (out.lev == level) {
          void *ptr;

          if (gel(alpha,1)!=gel(alpha,2)+lg(gel(alpha,1))) {
            ptr = complex_buf;
            memcpy(complex_buf, gel(alpha, 2),
                    lg(gel(alpha, 2)) * sizeof(ulong));
            memcpy(complex_buf + lg(gel(alpha, 2))*sizeof(ulong),
                    gel(alpha, 1),
                    lg(gel(alpha, 1)) * sizeof(ulong));
          }
          else
            ptr = (void *)gel(alpha, 2);
          MPI_Bcast(ptr,
              (lg(gel(alpha,1))+lg(gel(alpha,2)))*sizeof(ulong),
              MPI_CHAR, level, MPI_COMM_WORLD);
          aalpha = gsqrt(aalpha, prec);
        }
        else {
          GEN real;
          int n;

          alpha = cgetg(3, t_COMPLEX);
          gel(alpha, 2) = real = cgetg(2*prec, t_REAL);
          MPI_Bcast(real,2*prec*sizeof(ulong),MPI_CHAR, out.lev,
                      MPI_COMM_WORLD);
          gel(alpha, 1) = real + lg(real);
          n = lg(real) + lg(gel(alpha, 1));
          if (n < 2*prec) {
            int i;
```

```
          /* Clean up stack, should probably use stackdummy */
          real += n;
          settyp(real, t_VEC);
          setlg(real, 2*prec-n);
          for (i = 1; i < 2*prec - n; i++)
            gel(real, i) = gen_0;
        }
        imin = 0;
        aalpha = gabs(alpha, prec);
      }
    }
#else
    if (level) {
      int c;
      GEN atmp;

      writeGEN(alpha, filemaster);
      fflush(filemaster);
      atmp = readobj(filemaster, &c);
      if (!gequal(atmp, alpha)) {
          imin = 0;
          alpha = atmp;
          aalpha = gabs(alpha, prec);
      }
      else
          aalpha = gsqrt(aalpha, prec);
/*printf("r/%d:%p\n", c, alpha); */
/*printf("r:%d:%p\n", c, aalpha); */
    }
    else {
      aalpha = gsqrt(aalpha, prec);
      for (i = 1; i < np; i++) {
        int c;
        GEN abstmp;

        genpeers[i] = readobj(filepeers[i], &c);
/*printf("r+%d:%p::%p\n", c, genpeers[i], filepeers[i]); */
/*printf("%08x %08x %08x %08x\n", ((unsigned int *)genpeers[i])[0], ((unsig
        abstmp = gabs(genpeers[i], prec);
/*print(mkvec(abstmp)); */
        if (gcmp(abstmp, aalpha) < 0) {
          imin = 0;
          alpha = genpeers[i];
```

```
            aalpha = abstmp;
          }
/*printf("r*%d:%p\n", c, genpeers[i]); */
        }
        for (i = 1; i < np; i++) {
          writeGEN(alpha, filepeers[i]);
          fflush(filepeers[i]);
        }
      }
#endif
    }
/*if (!level) {print(mkvec(aalpha));print(mkvec(alpha));} */
  }
  else {
    aalpha = gabs(alpha, prec);
  }
  /*
   * s = abs(alpha)/alpha, cioè l'inverso del segno di \alpha.
   */
/*print(mkvec(mkvec3(stoi(level), alpha, aalpha))); */
  s = gdiv(aalpha, alpha);
  alpharen = gdiv(alpha, gsqrt(aalpha, prec));
  dom2 = gel(dom, 2);
  dom3 = gel(dom, 3);
  dom34= gel(dom3,4);
  /*
   * Vedere il commento di "conto" sopra per una giustifica
   * della formula.
   */
  /* s/(|\beta|+1/|\beta|)=\alpha/(|\beta|(1+|\alpha|)) */
  coeff = gdiv(alpharen, gadd(aalpha, uno));
  coeff3 = ginv(aalpha);
  coeff2 = gadd(aalpha, coeff3);
  /* Divisione per -2: */
  coeff2[1] = evalsigne(-1) | evalexpo(expo(coeff2)-1);
  coeff3 = gsqr(gsub(aalpha, coeff3));
  /* Divisione per -4: */
  coeff3[1] = evalsigne(-1) | evalexpo(expo(coeff3)-2);
  /* +1 */
#define FORMULA gmul(coeff, gsub(gadd(uno, divs), \
            gsqrt(gadd(gsqr(gadd(divs, coeff2)), coeff3), prec)))
  if (dom2 != gen_0) {
    lfuncs = glength(dom2);
    /*(void)timer(); */
    f = cgetg(lg(dom2), t_VEC);
    precdl++;
    divs = gmul(gel(dom2, 1), s);
    f1 = FORMULA;
    if (!valp(f1)) {
      gel(f1, 2) = gen_0;
      f1 = normalize(f1);
    }
    gel(f1, 2) = gabs(gel(f1, 2), prec);
    precdl--;
    gel(f, 1) = f1;
    {
      long i;

      for (i = 2; i <= lfuncs; i++) {
        divs = gmul(gel(dom2, i), s);
        gel(f, i) = FORMULA;
      }
    }
    /*series += timer(); */
    lbreak();
  }
  zz = cgetg(modl+1, t_VEC);
#if WITH_FORK
  sprintf(name, "/gp.%d", getpid());
  fd = shm_open(name, O_RDWR|O_CREAT|O_EXCL, 0x600);
  ftruncate(fd,
      (sizeof(GEN)+avma-bot+getpagesize()-1)&~(getpagesize()-1));
  mem = mmap(NULL, sizeof(GEN)+avma-bot, PROT_READ|PROT_WRITE,
          MAP_SHARED, fd, 0);
  zzp = mem + avma - bot;
  mid = modl/2;
  close(fd);
  switch(fork()) {
      case -1:
        pari_err(user, "fork() failed");
        exit(1);
      case 0:
        /* child */
        avma += (pari_sp)mem - bot;
        bot = (pari_sp)mem;
        zz = cgetg(modl - mid + 1, t_VEC);
        *zzp = zz;
        {
          long i;

          for (i = mid+1; i <= modl; ++i) {
            if (i == imin || gel(dom1, i) == gen_0) {
              /* Destruct the points, but keep 1/256th
               * for estimation */
              if (destructive && (iter & 255))
                gel(zz, i) = gen_0;
              else
                gel(zz, i) = alpharen;
            }
            else {
              divs = gmul(gel(dom1, i), s);
              gel(zz, i) = FORMULA;
            }
          }
        }
        exit(0);
      default:
#else
#define mid       modl
#endif
        {
          long i;

          /*(void)timer(); */
          for (i = 1; i <= mid; ++i) {
            /* Destruct the points, but keep 1/256th
             * for estimation */
            if (i == imin) {
              if (destructive && (iter & 255))
                gel(zz, i) = gen_0;
              else
                gel(zz, i) = alpharen;
            }
            else if (gel(dom1, i) == gen_0)
              gel(zz, i) = gen_0;
            else {
              divs = gmul(gel(dom1, i), s);
              gel(zz, i) = FORMULA;
            }
          }
          /*points += timer(); */
        }
#if WITH_FORK
  }
  wait(NULL);
  zz2 = *zzp;
  {
    int i;

    for (i = mid+1; i <= modl; i++)
      gel(zz, i) = gcopy(gel(zz2, i - mid));
  }
  munmap(mem, avma-bot);
  shm_unlink(name);
#endif
  if (!level)
    gel(zz, 1) = sgn(gel(zz, 1), prec);
  if (level == np - 1)
    gel(zz, modl) = sgn(gel(zz, modl), prec);
  rdom = cgetg(4, t_VEC);
  gel(rdom, 1) = zz;
  gel(rdom, 2) = dom2 == gen_0 ? gen_0 : f;
  gel(rdom, 3) = dom3;
  return rdom;
}

void
writemat(char *file, GEN var)      /* void */
{
  long l1;
  GEN s;

  unlink(file);
  if (typ(gel(var, 1)) == t_VEC)
    var = gcopy(gel(var, 1));
  l1 = glength(var);
  {
    long i;
    long l2;
    GEN p3;         /* vec */

    for (i = 1; i <= l1; i++)
    {
      s = gtovec(Str(mkvec3(greal(gel(var, i)),
                        strtoGENstr(","), gimag(gel(var, i)))));
      l2 = glength(s);
      {
        long i;
        GEN p4, p5;

        p3 = cgetg(l2+1, t_VEC);
        for (i = 1; i <= l2; ++i)
        {
          if (gequal(gel(s, i), strtoGENstr("␣")))
            p4 = strtoGENstr("");
          else
          {
            if (gequal(gel(s, i), strtoGENstr("E")))
              p5 = strtoGENstr("e");
            else
              p5 = gcopy(gel(s, i));
            p4 = p5;
          }
          gel(p3, i) = p4;
        }
      }
      write0(file, mkvec(concat(p3, NULL)));
    }
  }
  return;
}

GEN
taylorcoeffs(GEN dom)
{
    GEN f, t, z;
    ulong zn;

    /*
     * Con q=e^{2i\pi z/\lambda}, u=(ti-q)/(ti+q).
     * se f è centrata in -1, la variabile è 2ti/(ti+q)
     * se f è centrata in +1, la variabile è -2q/(ti+q)
     */
    f = gmael(dom, 2, glength(gel(dom, 2)));
    t = gmael(dom, 3, 2);
    zn = gvar(f);
    z = pol_x(zn);
    f = gsubst(f, zn, gdiv(gmulsg(-2, z), gadd(mulcxI(t), z)));

    return f;
}

void
writegnuplot(char *file, GEN dom)          /* void */
{
  char *str;
  char *cmd = "sed␣-i␣-e␣'s/[][()]//g'␣-e␣'s/,␣*/\\n/g'"          \
              "␣-e␣'s/␣E/e/g'␣-e␣'s/␣-␣/,␣-/g'␣-e␣'s/␣+␣/,␣/g'"\
              "␣-e␣'s/\\*I$//'␣";

  if (typ(gel(dom, 1)) == t_VEC)
    dom = taylorcoeffs(dom);
  unlink(file);
  write0(file, mkvec(dom));
  str = pari_malloc(strlen(cmd) + strlen(file) + 1);
  sprintf(str, "%s%s", cmd, file);
  system(str);
  free(str);
  return;
}

void
wg(long k, long prec)      /* void */
{
  long l1;
  GEN p2;          /* vec */
  char buf[100];

  l1 = glength(gel(f, k+1)) + 1;
  {
    long i;

    p2 = cgetg(l1, t_VEC);
    for (i = 1; i < l1; ++i)
      gel(p2, i) = gabs(polcoeff0(gel(f, k+1), i - 1, -1), prec);
  }
  /* doms[1][3][3] è limitato a 42 caratteri, k a 20 (per 64
   * bits) quindi buf ha una dimensione sufficiente */
  sprintf(buf, "coeff%ld-%s", k, GSTR(gmael3(doms, 1, 3, 3)));
  writegnuplot(buf, p2);
  return;
}

#ifndef MPI
int
addpeers(struct peer *pptr, int n, struct sockaddr_in *in)
{
  int ret = -1, i;
  static pthread_mutex_t peers_mutex = PTHREAD_MUTEX_INITIALIZER;

/*printf("%d: %d\n", ntohs(peers[0].p_addr.sin_port), n); */
  while (n--) {
/*printf("%d\n", ntohs(pptr->p_addr.sin_port)); */
    if (pptr->p_magic == RED_MAGIC) {
      in_addr_t curaddr_min = pptr->p_addr_min;

      pthread_mutex_lock(&peers_mutex);
      for (i = 0; i < parallel; i++) {
        if ((peers[i].p_addr_min == curaddr_min &&
            peers[i].p_addr.sin_port == pptr->p_addr.sin_port) ||
                !peers[i].p_addr_min)
          break;
      }
      if (!i) {
        pthread_mutex_unlock(&peers_mutex);
        in = NULL;
        pptr++;
        continue;
      }
      if (i < parallel) {
        if(!peers[i].p_addr_min) {
/*printf("r%d: (%08lx, %08lx, %d)\n", ntohs(peers[0].p_addr.sin_port), (ulc
/*if (in) printf("in: %08lx\n", (ulong)ntohl(in->sin_addr.s_addr)); */
          peers[i].p_magic = RED_MAGIC;
          /*
           * There is an implicit assumption on the topology of
           * the network: that the IP address with which an host
           * can be contacted is the same for all other hosts.
           */
          if (in) {
            peers[i].p_addr.sin_family  = in->sin_family;
            peers[i].p_addr.sin_addr.s_addr=in->sin_addr.s_addr;
            peers[i].p_addr.sin_port      = pptr->p_addr.sin_port;
          }
          else
            memcpy(&peers[i].p_addr, &pptr->p_addr,
                    sizeof peers[i].p_addr);
          peers[i].p_addr_min              = curaddr_min;
          peers[i].p_contacted             = 0;
          np++;
        }
        if (in)
          ret = i;
      }
      pthread_mutex_unlock(&peers_mutex);
      in = NULL;
    }
    pptr++;
  }
/*printf("%d:", ntohs(peers[0].p_addr.sin_port)); */
/*for (i = 0; i < np; i++) printf(" %d", ntohs(peers[i].p_addr.sin_port));
/*printf("\n"); */
  return ret;
}

int
recvpeers(int sock, struct sockaddr_in *in)
{
    struct peer pbuf[10];
    int nr, nb, ret = -1;
    long size;

    nb = 0;
    if (read(sock, &size, sizeof size) != sizeof size)
      return -1;
    while (size > 0 &&
            (nr = read(sock, ((char *)pbuf)+nb, sizeof pbuf-nb))
                > 0) {
      nb += nr;
      if (nb >= sizeof(*pbuf)) {
        if (ret == -1)
          ret = addpeers(pbuf, nb / sizeof(*pbuf), in);
        else
          addpeers(pbuf, nb / sizeof(*pbuf), in);
        memcpy(pbuf, pbuf + nb / sizeof *pbuf,
                nb % sizeof *pbuf);
        size -= nb / sizeof *pbuf;
        nb %= sizeof(*pbuf);
        in = NULL;
      }
    }
    return ret;
}

void *
thread_receiver(void *arg)
{
  int sockt = *(int *)arg;
  struct sockaddr_in in;
  int socklen, i;

  while (1) {
    int sock;

    socklen = sizeof in;
    sock = accept(sockt, (struct sockaddr*)&in, (void*)&socklen);
    if (sock >= 0) {
      uint32_t magic;

      write(sock, &state, sizeof state);
      if (read(sock, &magic, sizeof magic) == sizeof magic) {
        if (state == RED_MAGIC &&
                (magic == RED_MAGIC || magic == RED_MAGIC2)) {
/*printf("t"); */
          i = recvpeers(sock, &in);
          if (magic == RED_MAGIC) {
            write(sock, &np, sizeof np);
            write(sock, peers, np * sizeof(*peers));
            if (i > -1)
              peers[i].p_contacted = np;
          }
        }
        else if (magic == RED_MAGIC2 &&
                state == RED_MAGIC2 && level) {
/*printf("e%d\n", level); */
          sockmaster = sock;
          filemaster = fdopen(sock, "a+");
```

```
            pthread_exit(&sock);
          }
        }
        close(sock);
      }
    }
    /* NOT REACHED */
    return NULL;
}

void
broad(void)
{
    int sockd, sockt, sockm;
    int one = 1;
    struct sockaddr_in in, out;
    struct ifconf ifc;
    int i, n;
    int socklen;
    int min_addr = 0;
    char buf[1000];
    pthread_t child;
    void *sighandler;

    if (parallel < 2) {
      np = parallel = 1;
      return;
    }
    sighandler = signal(SIGPIPE, SIG_IGN);
    state = RED_MAGIC;
    sockd = socket(PF_INET, SOCK_DGRAM, 0);
    ifc.ifc_len = sizeof buf;
    ifc.ifc_buf = buf;
    ioctl(sockd, SIOCGIFCONF, &ifc);
    close(sockd);
    n = 4;
    for (i = 0; i < ifc.ifc_len/sizeof *ifc.ifc_req; i++) {
        int cur_addr = ntohl(((struct sockaddr_in *)
                        &ifc.ifc_req[i].ifr_addr)->
                            sin_addr.s_addr);

        if ((cur_addr >> 24) != IN_LOOPBACKNET) {
          if (min_addr > cur_addr)
```

```
            min_addr = cur_addr;
        }
    }
    sockd = socket(PF_INET, SOCK_DGRAM, 0);
    fcntl(sockd, F_SETFL, fcntl(sockd, F_GETFL) | O_NONBLOCK);
    out.sin_family = AF_INET;
    out.sin_addr.s_addr = INADDR_BROADCAST;
    i = 0;
    do
      out.sin_port = htons(RED_PORT+i);
    while (bind(sockd, (struct sockaddr *)&out, sizeof out)<0 &&
            i++ < 3);
    setsockopt(sockd, SOL_SOCKET, SO_BROADCAST, &one,sizeof one);
    sockt = socket(PF_INET, SOCK_STREAM, 0);
    in.sin_family = AF_INET;
    in.sin_addr.s_addr = INADDR_ANY;
    in.sin_port = htons(0);
    bind(sockt, (struct sockaddr *)&in, sizeof in);
    listen(sockt, 0);
    socklen = sizeof in;
    getsockname(sockt, (struct sockaddr *)&in, (void*)&socklen);
    srand48(time(NULL) + 5*getpid());
    peers = calloc(sizeof(struct peer), parallel);
    np = 1;
    peers[0].p_magic                   = RED_MAGIC;
    peers[0].p_addr.sin_family         = in.sin_family;
    peers[0].p_addr.sin_addr.s_addr    = htonl(min_addr);
    peers[0].p_addr.sin_port           = in.sin_port;
    peers[0].p_addr_min                = htonl(min_addr);
/*printf("p%d: %d\n", getpid(), htons(in.sin_port)); */
    if (pthread_create(&child, NULL, &thread_receiver, &sockt)) {
      perror("pthread_create");
      exit(1);
    }
    if (sockt > sockd)
      sockm = sockt + 1;
    else
      sockm = sockd + 1;
    while (np < parallel) {
      struct timeval    timeout;
      fd_set            set;

/*printf("b: %ld, %d\n", np, parallel); */
```

```
      timeout.tv_usec  = 500000 + (lrand48() >> 11);
      timeout.tv_sec   = timeout.tv_usec / 1000000;
      timeout.tv_usec -= timeout.tv_sec  * 1000000;
      FD_ZERO(&set);
      FD_SET(sockt, &set);
      FD_SET(sockd, &set);
      switch (select(sockm, &set, NULL, NULL, &timeout)) {
        case -1:
          /* Error */
#ifdef ERESTART
          if (errno == ERESTART)
            break;
#endif
          fprintf(stderr, "Problem while contacting peers.\n");
          exit(1);

        case 0:
          /* Timeout */
          for (i = 0; i < 3; i++) {
            int j;

            out.sin_addr.s_addr = INADDR_BROADCAST;
            out.sin_port = htons(RED_PORT + i);
            sendto(sockd, peers, sizeof(*peers), 0,
                   (struct sockaddr *)&out, sizeof out);
            for (j = 0; j < numother; j++) {
              if (otherhosts[j].sin_port) {
                if (!i) {
                  int sock;

                  sock = socket(PF_INET, SOCK_STREAM, 0);
                  if (!connect(sock,
                               (struct sockaddr *)&otherhosts[i],
                               sizeof otherhosts[i])) {
                    uint32_t magic;

                    write(sock, &state, sizeof state);
                    if (read(sock, &magic, sizeof magic) ==
                          sizeof magic &&
                        (magic==RED_MAGIC||magic==RED_MAGIC2)) {
                      if (magic == RED_MAGIC) {
                        write(sock, &np, sizeof np);
                        write(sock, peers, np * sizeof(*peers));
                    }
                    recvpeers(sock, &otherhosts[i]);
                  }
                }
                close(sock);
              }
            }
            else {
              out.sin_addr.s_addr =
                  otherhosts[j].sin_addr.s_addr;
              sendto(sockd, peers, sizeof(*peers), 0,
                     (struct sockaddr *)&out, sizeof out);
            }
          }
        }
        for (i = 1; i < np; i++) {
          if (peers[i].p_contacted < np) {
            int sock = socket(PF_INET, SOCK_STREAM, 0);

            if (!connect(sock,
                    (struct sockaddr *)&peers[i].p_addr,
                    sizeof peers[i].p_addr)) {
              uint32_t magic;

              write(sock, &state, sizeof state);
              if (read(sock, &magic, sizeof magic) ==
                      sizeof magic &&
                  (magic == RED_MAGIC || magic == RED_MAGIC2)) {
                if (magic == RED_MAGIC) {
                  write(sock, &np, sizeof np);
                  write(sock, peers, np * sizeof(*peers));
                }
/*printf("s"); */
                recvpeers(sock, NULL);
                peers[i].p_contacted = np;
              }
              else if (magic == RED_MAGIC2)
                peers[i].p_contacted = np;
            }
            close(sock);
          }
        }
        break;
        default:
          /* Someone is calling */
          if (FD_ISSET(sockd, &set)) {
            int nr;
            struct peer *pptr = (struct peer *)buf;

            socklen = sizeof in;
            nr = recvfrom(sockd, buf, sizeof buf, 0,
                    (struct sockaddr *)&in, (void*)&socklen);
            if (nr == sizeof(struct peer)) {
/*printf("d"); */
              addpeers(pptr, 1, &in);
            }
          }
          break;
      }
    }
    close(sockd);
/*{char *p = buf; p += sprintf(p, "%d: ", ntohs(peers[0].p_addr.sin_port));
    level = 0;
    for (i = 1; i < np; i++)
      level += ((ntohl(peers[i].p_addr_min) <
                  ntohl(peers[0].p_addr_min)) ||
                (peers[i].p_addr_min == peers[0].p_addr_min &&
                 ntohs(peers[i].p_addr.sin_port) <
                     ntohs(peers[0].p_addr.sin_port)));
/*printf("%d: l%d\n", ntohs(peers[0].p_addr.sin_port), level); */
    state = RED_MAGIC2;
    imaster = 0;
    if (level) {
      /* Inform "master" about the hosts. */
      int         sock, i;
      in_addr_t  addr_min=ntohl(peers[0].p_addr.sin_addr.s_addr);
      in_port_t  min = ntohs(peers[0].p_addr.sin_port);
      void        *child_ret;

      for (i = 1; i < np; i++) {
/*printf("M%d: act = (%08lx, %d), test = (%08lx, %d)\n", level, (ulong)addr
          if (ntohl(peers[i].p_addr_min) < addr_min ||
              (ntohl(peers[i].p_addr_min) == addr_min &&
               ntohs(peers[i].p_addr.sin_port) < min)) {
            addr_min = ntohl(peers[i].p_addr_min);
          min = ntohs(peers[i].p_addr.sin_port);
          imaster = i;
        }
      }
/*printf("master: %d, addr=(%d, %08lx, %d)\n", imaster, peers[imaster].p_ac
      while (1) {
        uint32_t magic;
        int rd;

        sock = socket(PF_INET, SOCK_STREAM, IPPROTO_IP);
/*printf("sock=%d\n", sock); */
        while (connect(sock,
                       (struct sockaddr *)&peers[imaster].p_addr,
                       sizeof peers[0].p_addr) == -1) {
          perror("connect");
          usleep(lrand48() >> 19);
        }
        write(sock, &state, sizeof state);
        rd = read(sock, &magic, sizeof magic);
        if (rd == sizeof magic) {
          if (magic == RED_MAGIC) {
            write(sock, &np, sizeof np);
            write(sock, peers, np * sizeof(*peers));
            close(sock);
          }
          else if (magic == RED_MAGIC2) {
            close(sock);
            break;
          }
          else {
            close(sock);
            usleep(lrand48() >> 19);
          }
        }
        else {
            close(sock);
            usleep(lrand48() >> 19);
        }
      }
      pthread_join(child, &child_ret);
    }
    else {
      int sock;
```

```
sockpeers = (int *)pari_malloc(np * sizeof(int));
filepeers = (FILE **)pari_malloc(np * sizeof(FILE *));
peerlevel = (int *)pari_malloc(np * sizeof(int));
for (i = 1; i < np; i++) {
  uint32_t magic;

  if (i == imaster)
    continue;
  sock = socket(PF_INET, SOCK_STREAM, 0), i;
  while (connect(sock,
               (struct sockaddr *)&peers[i].p_addr,
               sizeof peers[0].p_addr) == -1)
      usleep(lrand48() >> 19);
  write(sock, &state, sizeof state);
  if (read(sock, &magic, sizeof magic) == sizeof magic) {
    if (magic == RED_MAGIC) {
      write(sock, &np, sizeof np);
      write(sock, peers, np * sizeof(*peers));
      i--;
      close(sock);
    }
    else if (magic != RED_MAGIC2) {
      usleep(lrand48() >> 19);
      i--;
      close(sock);
    }
    else {
      sockpeers[i] = sock;
      filepeers[i] = fdopen(sockpeers[i], "a+");
    }
  }
  else {
    usleep(lrand48() >> 19);
    i--;
    close(sock);
  }
  close(-1);
}
for (i = 0; i < np; i++) {
  int pl = 0, j;

  for (j = 0; j < np; j++)
```

```
            pl += ((ntohl(peers[j].p_addr_min) <
                        ntohl(peers[i].p_addr_min)) ||
                      (peers[j].p_addr_min == peers[i].p_addr_min&&
                       ntohs(peers[j].p_addr.sin_port) <
                        ntohs(peers[i].p_addr.sin_port)));
/*printf("pl%d: %d\n", i, pl);*/
          peerlevel[pl] = i;
        }
/*for (i = 0; i < np; i++)
printf("Peer with level %d is %d with addr %08lx and port %d\n", i, peerle
    }
    signal(SIGPIPE, SIG_IGN);
}
#endif

void
usage(char *name, FILE *f)
{
  fprintf(f, "Usage: %s [options]\n", name);
  fprintf(f, "Where options are:\n");
  fprintf(f, "  -h           : get a little help\n");
  fprintf(f, "  -l <real>    : lambda\n");
  fprintf(f, "  -t <real>    : t\n");
  fprintf(f, "  -n <integer>: number of point of frontier\n");
  fprintf(f, "  -k <integer>: number of iterations\n");
  fprintf(f, "  -m <integer>: allocated memory\n");
  fprintf(f, "  -M <integer>: log2(allocated memory)\n");
  fprintf(f, "  -p <integer>: real precision\n");
  fprintf(f, "  -s <integer>: series precision\n");
  fprintf(f, "  -P <integer>: number of processes\n");
  fprintf(f, "  -f <file>    : restart from <file>\n");
  fprintf(f, "  -c <integer>: step of the computation");
          fprintf(f, " (useful with -f)\n");
  fprintf(f, "  -N           : be nasty (nice 0 instead of");
          fprintf(f, " 18)\n");
  fprintf(f, "  -D           : destructive on the border of U\n");
  fprintf(f, "  -u           : choose progressive succession");
          fprintf(f, " of alpha\n");
  fprintf(f, "  -r           : choose pseudo-random succession");
          fprintf(f, " of alpha\n");
}

int
```

```
main(int argc, char **argv)
{
    pari_sp ltop, limit;
    int i, optloop;
    char *lambda = "5.23";
    char *recup = NULL;
    GEN t = NULL, n = NULL;
    char *ts = NULL, *ns = NULL;
    char *dirname = 0;
    int prec = MEDDEFAULTPREC;
    unsigned long mem = 128 * 1024 * 1024;
    int dl = -1;
    unsigned long powof2 = 1;
    long log2ofpow = 0;
    int nice = 18, o;
#ifndef MPI
    struct hostent *he;
#endif
    struct timeval tvstart, tvstop;
    int randompoints = 0;

    umask(022);
#if 0
{
    char buf[1000];

    close(1);
    close(2);
    strcpy(buf, "out.");
    gethostname(buf+strlen(buf), sizeof buf-strlen(buf));
    sprintf(buf+strlen(buf), "%d", getpid());
    if (!open(buf, O_RDWR|O_CREAT|O_APPEND, 0666))
        open(buf, O_RDWR|O_CREAT|O_APPEND, 0666);
    memcpy(buf, "err", 3);
    open(buf, O_RDWR|O_CREAT|O_APPEND, 0666);
}
#endif
    gettimeofday(&tvstart, NULL);
    iter = 0;
    maxiter = (ulong)-1;
    optloop = 1;
#ifdef MPI
    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &np);
    parallel = np;
    MPI_Comm_rank(MPI_COMM_WORLD, &level);
/* printf("Pid %d, rank %d of %d\n", getpid(), level, np); */
#else
    numother = 0;
    otherhosts = malloc(argc * sizeof *otherhosts);
#endif
    while (optloop) {
#ifdef MPI
      switch(o=getopt(argc,argv,"hl:t:n:p:s:P:k:m:M:f:NDurc:"))
#else
      switch(o=getopt(argc,argv,"hl:t:n:p:s:P:k:m:M:f:NDuro:c:"))
#endif
      {
        case 'h':
            usage(argv[0], stdout);
            exit(0);

        case 'l':
            lambda = optarg;
            break;

        case 't':
            ts = optarg;
            break;

        case 'n':
            ns = optarg;
            break;

        case 'k':
            maxiter = strtol(optarg, NULL, 0);
            break;

        case 'm':
            { char *endptr;

            mem = strtol(optarg, &endptr, 0);
            if (endptr) {
              switch (*endptr) {
                case 'k': case 'K':
                  mem <<= 10;
```

```
          break;
        case 'm': case 'M':
          mem <<= 20;
          break;
        case 'g': case 'G':
          mem <<= 30;
          break;
      }
    }}
    break;

case 'M':
    mem = 1 << strtol(optarg, NULL, 0);
    break;

case 'p':
    prec = 3 + log(10) * strtol(optarg, NULL, 0) /
                           log(2) / BITS_IN_LONG;
    break;

case 's':
    dl = strtol(optarg, NULL, 0);
    break;

case 'P':
    parallel = strtol(optarg, NULL, 0);
    break;

case 'f':
    recup = optarg;
    break;

case 'N':
    nice = 0;
    break;

case 'D':
    destructive = 1;
    break;

case 'u':
case 'r':
    if (randompoints && randompoints != 1+(o == 'u')) {
```

```
                fprintf(stderr, "-u and -r are not compatible\n");
                exit(1);
            }
            randompoints = 1+(o == 'u');;
            break;

#ifndef MPI
        case 'o':
            {
                char *port = strchr(optarg, ':');

                if (port) {
                    *port++ = 0;
                }
                he = gethostbyname(optarg);
                memcpy(&otherhosts[numother++],
                        &he->h_addr_list[0],
                        sizeof(he->h_addr_list[0]));
                if (port) {
                    otherhosts[numother-1].sin_port =
                            htons(strtol(port, NULL, 0));
                }
                else {
                    otherhosts[numother-1].sin_port = htons(0);
                }
            }
            break;
#endif

        case 'c':
            iter = strtol(optarg, NULL, 0);
            break;

        case -1:
            optloop = 0;
            break;

        case '?':
        case ':':
            usage(argv[0], stderr);
            exit(1);
        }
    }
    setpriority(PRIO_PROCESS, 0, nice);
    pari_init(mem, 500000);
    (void)pol_x(fetch_user_var("y"));
    (void)pol_x(fetch_user_var("z"));
    (void)pol_x(fetch_user_var("t"));
    (void)pol_x(fetch_user_var("u"));
    (void)pol_x(fetch_user_var("q"));

    signal(SIGUSR1, (void*)&usr1handler);

#ifdef SIGHUP
    hdlr[SIGHUP] = signal(SIGHUP, (void*)&save_state);
#endif
#ifdef SIGINT
    hdlr[SIGINT] = signal(SIGINT, (void*)&save_state);
#endif
#ifdef SIGTERM
    hdlr[SIGTERM] = signal(SIGTERM, (void*)&save_state);
#endif
#ifdef SIGQUIT
    hdlr[SIGQUIT] = signal(SIGQUIT, (void*)&save_state);
#endif
#ifdef SIGIOT
    hdlr[SIGIOT] = signal(SIGIOT, (void*)&save_state);
#endif
#ifdef SIGBUS
    hdlr[SIGBUS] = signal(SIGBUS, (void*)&save_state);
#endif
#ifdef SIGFPE
    hdlr[SIGFPE] = signal(SIGFPE, (void*)&save_state);
#endif
#ifdef SIGXCPU
    hdlr[SIGXCPU] = signal(SIGXCPU, (void*)&save_state);
#endif
#ifdef SIGPWR
    hdlr[SIGPWR] = signal(SIGPWR, (void*)&save_state);
#endif
#ifdef SIGUSR2
    hdlr[SIGUSR2] = signal(SIGUSR2, (void*)&save_state);
#endif
    if (dl != -1) {
      precdl = dl;
    }
    if (ns) {
      n = strtoi(ns);
    }
    ltop = avma;
    limit = stack_lim(avma, 2);
    precreal = prec;
    if (ts) {
        t = strtor(ts, prec);
    }
    /*sleep(10);*/
#ifndef MPI
    broad();
    genpeers = (GEN *)pari_malloc(np * sizeof(GEN));
#else
    complex_buf = pari_malloc(2*(prec+1)*sizeof(ulong));
#endif
    if (recup) {
      char buf[1000];
      char *p;

      cur_dom = NULL;
      if (np > 1 && strlen(recup) < sizeof(buf)-10) {
        if ((p = strchr(recup, '@'))) {
          *p = '\0';
          p += 2;
          while (isdigit(*p)) {
              p++;
          }
          sprintf(buf, "%s@l%d%s", recup, level, p);
          if ((p = strchr(buf, '/'))) {
            *p = '\0';
            dirname = strdup(buf);
            *p = '/';
          }
          recup = buf;
        }
      }
      else
        if ((p = strchr(recup, '/'))) {
          *p = '\0';
          dirname = strdup(recup);
          *p = '/';
        }
        if (!cur_dom)
            cur_dom = gp_read_file(recup);
        if (typ(gel(cur_dom, 1)) == t_INT) {
          iter = itos(gel(cur_dom, 1));
          if (maxiter == (ulong)-1)
              maxiter = itos(gel(cur_dom, 2));
/*printf("Level %lu, %lu of %lu\n", level, iter, maxiter);*/
          cur_dom = gel(cur_dom, 3);
        }
        else {
          char *num;

          if (!iter && (num = strrchr(recup, '-'))) {
            log2ofpow = atoi(num);
            powof2 = 1 << -log2ofpow;
            iter = powof2 >> 1;
          }
        }
        gel(cur_dom, 3) = gclone(gel(cur_dom, 3));
        precdl = glength(gmael(cur_dom, 3, 4))+1;
        precreal = prec = lg(gmael(cur_dom, 3, 1));
        uno = gclone(real_1(prec));
        if (!n && glength(gel(cur_dom, 3)) < 6 && np == 1) {
            n = stoi(glength(gel(cur_dom, 1))-1);
        }
#ifdef MPI
        if (!level) c0 = lg(gel(cur_dom, 1)) - 1;
        MPI_Bcast(&c0, 1, MPI_LONG, 0, MPI_COMM_WORLD);
#else
        if (level)
          fread(&c0, sizeof c0, 1, filemaster);
        else
          for (i = 1; i < np; i++)
        fwrite(&c0, sizeof c0, 1, filepeers[peerlevel[i]]);
#endif
        chunk0 = c0 * np;
        /* Acts as an offset for the computations of the indices */
        c0++;
      }
      else {
        uno = gclone(real_1(prec));
        cur_dom = gerepilecopy(ltop,
                domain(strtor(lambda, prec), t, n, prec));
```

```
    }
/*printf("l%d: %d\n", level, glength(gel(cur_dom, 1))); */
/*printf("al%d %08x %08x %08x %08x\n", level, ((unsigned int *)gel(cur_dom,
/*printf("f%d\n", level); */
    if (dirname)
      chdir(dirname);
    else
    {
        char *str = GSTR(gmael(cur_dom, 3, 3)), *str2;
        char buf[1000];

        str2 = strchr(str, '-');
        if (str2) {
            *str2 = '\0';
            if (np > 1) {
              sprintf(buf, "%s@l%d", str, level);
              str = buf;
            }
            mkdir(str, 0777);
            chdir(str);
            *str2 = '-';
        }
    }
    {
        char buf[1024];

        /*sprintf(buf, "out-%d.", level);
        gethostname(buf+strlen(buf),sizeof(buf)-strlen(buf)-1);*/
        strcpy(buf, "out");
        (void)freopen(buf, "a", stdout);
        memcpy(buf, "err", 3);
        (void)freopen(buf, "a", stderr);
    }
    if (iter) {
      if (2 * iter != powof2) {
        powof2 = 1;
        log2ofpow = 0;
        while (powof2 <= iter) {
          powof2 <<= 1;
          log2ofpow--;
        }
      }
    }
    else {
      iter = 1;
    }
    if (!n) {
      n = gmael(cur_dom, 3, lg(gel(cur_dom, 3))-1);
    }
    if (np == 1)
      last_peer = 1;
    else
      last_peer = (itos(n) + 1 - chunk0) % (np - 1);
    if (randompoints) {
      GEN dom1 = gel(cur_dom, 1);
      ulong l = lg(dom1) - 1, i;

      lgthes = (long *)pari_malloc(np*sizeof(long));
#ifdef MPI
      MPI_Allgather(&l, 1, MPI_LONG, lgthes, 1,
                    MPI_LONG, MPI_COMM_WORLD);
#else
    if (!level) {
      lgthes[0] = l;
      /* Gather */
      for (i = 1; i < np; i++)
        fread (lgthes + i,  1, sizeof(long), filepeer[i]);
      /* Broadcast */
      for (i = 1; i < np; i++)
        fwrite(lgthes    , np, sizeof(long), filepeer[i]);
    }
    else {
      /* Gather */
        fwrite(&l        ,  1, sizeof(long), filemaster);
      /* Broadcast */
        fread (lgthes    , np, sizeof(long), filemaster);
    }
#endif
/*  printf("[");for(i=0;i<np;i++)printf(" %d%c",lgthes[i],i<np-1?',':']');p
      for (i = 1; i < np; i++)
        lgthes[i] += lgthes[i-1];
      globaln = lgthes[np-1]-1;
      if (randompoints == 1) {
        if (np == 1)
          idxincr = 2;
        else
```

```
          idxincr = lgthes[1];
        do
            idxincr++;
        while (cgcd(idxincr, globaln)!=1);
        if (idxincr > globaln)
          idxincr -= globaln;
      }
      else
        idxincr = 1;
      curidx = itos(lift(gmulsg(iter,
                              gmodulss(idxincr, globaln))));
/* printf("n = %ld, idxincr = %d, iter = %ld, curidx = %ld\n", globaln, idx
    }
    else
      lgthes = NULL;
    if (maxiter == (ulong)-1) {
      if (lgthes)
        maxiter = itos(n)-1;
      else
        maxiter = itos(n)/4;
    }
    if (iter == 1)
      writebin(GSTR(concat(gmael(cur_dom, 3, 3),
                              strtoGENstr("-0"))), cur_dom);
    for (; iter <= maxiter; iter++) {
      cur_dom = stepkoebe(cur_dom, NULL, prec);
      if (avma < limit) {
        GEN dom = cur_dom;

        /*
         * So that save_state does not try to save an invalid
         * cur_dom.
         */
        cur_dom = NULL;
        cur_dom = gerepilecopy(ltop, dom);
      }
      if (iter == powof2) {
        char *s;
        time_t t;

        writebin(GSTR(concat(gmael(cur_dom,3,3),
                              stoi(--log2ofpow))), cur_dom);
        time(&t);
```

```
          s = ctime(&t);
          if (s[strlen(s)-1] == '\n')
              s[strlen(s)-1] = '\0';
          printf("%s %ld\n", s, -log2ofpow);
          fflush(stdout);
          powof2 <<= 1;
        }
      }
      if (np > 1) {
        char *filename;

        filename = GSTR(concat(gmael(cur_dom, 3, 3),
                        strtoGENstr("-prefinal")));
        writebin(filename, cur_dom);
      }
      if (!level) {
        char *filename;
        GEN dompeer;

        if (maxiter > 1) {
            cur_dom = gerepilecopy(ltop, cur_dom);
        }
        for (i = 1; i < np; i++) {
#ifdef MPI
          MPI_Status status;
          void *buf;
          int size;

          MPI_Probe(i, 0, MPI_COMM_WORLD, &status);
          MPI_Get_count(&status, MPI_CHAR, &size);
          buf = pari_malloc(size);
          MPI_Recv(buf, size, MPI_CHAR, i, 0, MPI_COMM_WORLD,
                   MPI_STATUS_IGNORE);
          dompeer = bin_copy(buf);
#else
          int c;

          dompeer = readobj(filepeers[peerlevel[i]], &c);
#endif
          gel(cur_dom, 1) = concat(gel(cur_dom, 1), dompeer);
          cur_dom = gerepilecopy(ltop, cur_dom);
        }
        filename = GSTR(concat(gmael(cur_dom, 3, 3),
                    strtoGENstr("-final")));
        writebin(filename, cur_dom);
    }
    else {
#ifdef MPI
        GENbin *buf;
        int size;

        buf = copy_bin(gel(cur_dom, 1));
        size = sizeof(GENbin) + buf->len*sizeof(ulong);
        MPI_Send(buf, size, MPI_CHAR, 0, 0, MPI_COMM_WORLD);
        free(buf);
#else
        writeGEN(gel(cur_dom, 1), filemaster);
#endif
    }
#ifdef MPI
    MPI_Finalize();
/*printf("series: %7llu.%03llus\npoints: %7llu.%03llus\n", series/1000, se
#endif
/*printf("Coucou%d\n", level); */
    gettimeofday(&tvstop, NULL);
 printf("Time: %ld\n",
        1000*(tvstop.tv_sec - tvstart.tv_sec) +
            (tvstop.tv_usec-tvstart.tv_usec)/1000);
    return 0;
}

/*
:se makeprg=make\ debug
 vi:aw
*/
```

Second version of the program

```
#ifdef __CYGWIN__
typedef unsigned long ulong;
#endif
#include <pari/pari.h>
#include <pari/paripriv.h>
#include <fcntl.h>
#include <time.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/select.h>
#include <errno.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <ctype.h>
#include <sys/resource.h>
#include <unistd.h>
#ifdef MPI
#include <mpi.h>
#endif

/*
 * Algorithme du premier cas (avec la racine carrÃ©e).
 */
#define VIEUX

#define NPOINTS 65536
#define NSAVE     1000

static void *hdlr[NSIG];

static GEN uno;
static GEN cur_dom;
static ulong iter;
static int pending_sig;
#ifdef MPI
static GEN alpha_buf, complex_buf;
#endif
#ifndef MPI
```

```
const
#endif
static int np = 1, level = 0;

static unsigned long long tseries = 0, tpoints = 0;

static void lbreak(void) {}

/*
GP; install("sgn","D0,G,p","sgn","./reduced.gp.so");
GP; install("domain","Gp","domain","./reduced.gp.so");
GP; install("stepkoebe","Gp","stepkoebe","./reduced.gp.so");
GP; install("init_reduced","v","init_reduced","./reduced.gp.so");
*/
GEN sgn(GEN z, long prec);
GEN domain(GEN lambda, long prec);
GEN stepkoebe(GEN dom, long prec);
void init_reduced(void);
/*End of prototype*/

void
init_reduced(void)          /* void */
{
  return;
}

void
usr1handler(void)
{
  char *s;
  time_t t;

  time(&t);
  s = ctime(&t);
  if (s[strlen(s)-1] == '\n')
      s[strlen(s)-1] = '\0';
  printf("%s i=%lu\n", s, iter);
  fflush(stdout);
}

void
save_state(int sig)
{
  if (cur_dom) {
    writebin("emergency", mkvec2(stoi(iter), cur_dom));
    printf("Done␣(%lu).\n", iter);
    pending_sig = 0;
  }
  else {
    pending_sig = sig;
    if (!cur_dom)
      printf("Sorry:␣garbage␣collecting␣(%lu).\n", iter);
  }
  fflush(stdout);
  if (sig != SIGUSR2) {
    sigset_t set;

    signal(sig, hdlr[sig]);
    sigemptyset(&set);
    sigaddset(&set, sig);
    sigprocmask(SIG_UNBLOCK, &set, NULL);
#ifdef MPI
    MPI_Barrier(MPI_COMM_WORLD);
#endif
    kill(getpid(), sig);
    /* Segnale non fatale ! */
    signal(sig, (void *)&save_state);
  }
  else
    signal(sig, (void*)&save_state);
}

GEN
sgn(GEN z, long prec)
{
  if (!z || z == gen_0)
      return gen_0;
  return gdiv(z, gabs(z, prec));
}

GEN
domain(GEN lambda, long prec)      /* vec */
{
    pari_sp av = avma;
    GEN v, name, namec;
    GEN z = pol_x(fetch_user_var("z"));
    GEN q = pol_x(fetch_user_var("q"));
    GEN u = pol_x(fetch_user_var("u"));
    GEN dom, vecf;          /* vec */
    GEN A1, A2, c, twopioverlambda, zeta, tmp1, tmp2;
    GEN hq1, hqi, signAbarsq, monz;
    GEN incr;
    ulong i, lg1;

    dom = cgetg(4, t_VEC);
    if (level == np - 1)
      v = cgetg(1+NPOINTS-level*(NPOINTS/np), t_VEC);
    else
      v = cgetg(1+NPOINTS/np, t_VEC);
    lg1 = lg(v)-1;
    gel(dom, 1) = v;
    incr = gexp(mulcxI(gdivgs(constpi(prec), -NPOINTS/2)), prec);
    twopioverlambda = gmul2n(gdiv(constpi(prec), lambda), 1);
    c = gch(twopioverlambda, prec);
    A1 = gexp(mulcxI(gmul2n(twopioverlambda, 1)), prec);
    A1 = gsub(uno, gmul(c, A1));
    signAbarsq = gconj(sgn(gsqr(A1), prec));
    A2 = gneg(gimag(A1));
    A1 = greal(A1);
    zeta = gdivgs(constpi(prec), -NPOINTS/2);
    zeta = gexp(mulcxI(gmulsg(level*(NPOINTS/np), zeta)), prec);
    zeta = gneg(gmul2n(zeta, -1));
    for (i = 1; i <= lg1; i++)
    {
      gel(v, i) = zeta;
      zeta = gmul(zeta, incr);
    }
    vecf = cgetg(2, t_VEC);
    gel(dom, 2) = vecf;
    i = 1;
    gel(vecf, i++) = gcopy(u);
    if (i != lg(vecf))
      pari_err(talker,
        "Improper vecf length in domain (%d instead of %d)",
        lg(vecf), i);
    gel(dom, 2) = vecf;
    if (1)
      namec = strtoGENstr("-0");
    else
```

```
{
  GEN t = gen_1;

  namec = concat(strtoGENstr("-"), GENtoGENstr(t));
  if (typ(t) == t_REAL)
  {
    char *namep = GSTR(namec);
    long i;

    for (i = glength(namec); i > 0; i--)
      if (namep[i-1] != '0' && i <= 21)
      {
        namep[i] = '\0';
        namec = strtoGENstr(namep);
        break;
      }
  }
}
name = GENtoGENstr(lambda);
if (typ(lambda) == t_REAL)
{
  char *namep = GSTR(name);
  long i;

  for (i = glength(name); i > 0; i--)
    if (namep[i-1] != '0' && i <= 20)
    {
      if (namep[i-1] == '.') {
        if (i == 1)
          namep = "0"; /* unlikely */
        else
          namep[i-1] = '\0';
      }
      else
        namep[i] = '\0';
      name = strtoGENstr(namep);
      break;
    }
}
name = concat(name, namec);

vecf = cgetg(9, t_VEC);
v = cgetg(np+2, t_VECSMALL);
```

```
    v[1] = 0;
    for (i = 1; i < np; i++)
      v[i+1] = v[i] + NPOINTS / np;
    v[np+1] = NPOINTS;
    i = 1;
    gel(vecf, i++) = lambda;
    gel(vecf, i++) = gen_0;
    gel(vecf, i++) = name;
    gel(vecf, i++) = A1;
    gel(vecf, i++) = A2;
    gel(vecf, i++) = gmul(c, A2);
    gel(vecf, i++) = twopioverlambda;
    gel(vecf, i++) = v;
    if (i != lg(vecf))
      pari_err(talker,
        "Improper third component length (%d instead of %d).\n",
        lg(vecf), i);
    /* Create third element (never copied) on the heap */
    vecf = gclone(vecf);
    gel(dom, 3) = gen_0;
    dom = gerepileupto(av, dom);
    gel(dom, 3) = vecf;

    return dom;
}

long
minc(GEN L)
{
  GEN loicmin, loictmp;
  long iloicmin;
  long l1;

  loicmin = gnorm(gel(L, 1));
  iloicmin = 1;
  l1 = glength(L);
  {
    int i;
    for (i = 2; i <= l1; i++)
    {
      loictmp = gnorm(gel(L, i));
      if (gcmp(loictmp, loicmin) < 0)
      {
          iloicmin = i;
          loicmin = loictmp;
        }
      }
    }
    return iloicmin;
}

GEN
glogoff(GEN z, GEN off, long prec)
{
  GEN logz;

  switch(typ(z))
  {
    case t_INT:
    case t_FRAC:
    case t_REAL:
    case t_COMPLEX:
      logz = glog(z, prec);
      if (typ(logz) == t_COMPLEX)
      {
        GEN tmpi = gadd(gel(logz, 2), off);
        if (signe(off) > 0 && gcmp(tmpi, Pi2n(-1, prec)) > 0)
          tmpi = gsub(tmpi, constpi(prec));
        else if (gcmp(tmpi, gneg_i(Pi2n(-1, prec))) < 0)
          tmpi = gadd(tmpi, constpi(prec));
        gel(logz, 2) = tmpi;
        return logz;
      }
      else
      {
        GEN tmpz = cgetg(3, t_COMPLEX);
        gel(tmpz, 1) = logz;
        gel(tmpz, 2) = off;
        return tmpz;
      }
      /* NOT REACHED */
      break;

    case t_POL:
    case t_RFRAC:
    case t_SER:
      logz = glog(z, prec);
      if (valp(logz) > 0)
        logz = gadd(logz, mulcxI(off));
      else {
        GEN coef0;

        coef0 = gel(logz, 2-valp(logz));
        if (typ(coef0) == t_COMPLEX)
        {
          GEN tmpi = gadd(gel(coef0, 2), off);
          if (signe(off) > 0 && gcmp(tmpi, Pi2n(-1, prec)) > 0)
            tmpi = gsub(tmpi, constpi(prec));
          else if (gcmp(tmpi, gneg_i(Pi2n(-1, prec))) < 0)
            tmpi = gadd(tmpi, constpi(prec));
          gel(coef0, 2) = tmpi;
        }
        else
        {
          GEN tmpz = cgetg(3, t_COMPLEX);

          gel(tmpz, 1) = coef0;
          gel(tmpz, 2) = off;
          gel(logz, 2-valp(logz)) = tmpz;
        }
      }
      return logz;

    default:
      pari_err(typeer,"glogoff");
  }
  /* NOT REACHED */
  return gnil;

}

GEN
stepkoebe(GEN dom, long prec)      /* vec */
{
  GEN zz, f=0, f1=0, aalpha, lognalpha, calpha;
  static ulong powof2 = 1;
  static GEN pol = NULL;
  GEN dom1 = gel(dom, 1), dom2;
#ifdef VIEUX
```

```
  GEN coeff2, coeff3;
#endif
  long i, lg1 = lg(dom1), idx;
  GEN alpha;
  GEN rdom;          /* vec */
  GEN tmp1, tmp2;
  int peer_idx;
#ifdef MPI
  int mask = sigblock(-1);
#endif

#ifdef MPI
  {
      int tmp_sig = -1;

      /* In at least one implementation, MPI_Allreduce has
       * no effect if tmp_sig==pending_sig */
      MPI_Allreduce(&pending_sig,&tmp_sig,1, MPI_INT,
              MPI_MAX, MPI_COMM_WORLD);
      pending_sig = tmp_sig;
  }
#endif
  if (pending_sig)
    save_state(pending_sig);
#ifdef MPI
  sigsetmask(mask);
#endif
  while (powof2 < iter) powof2 *= 2;
  /*idx = minc(dom1);*/
  idx = (NPOINTS*(2*iter - 1 - powof2))/powof2;
  if (np > 1)
    peer_idx = idx / (NPOINTS / np);
  else
    peer_idx = 0;
  if (peer_idx == level) {
    int decr = 2*iter - 2 - powof2;

    lg1--;
    /* Cancelled points */
    decr -= gmael(dom, 3, 8)[1+level] / (NPOINTS/powof2);
    idx -= decr + gmael(dom, 3, 8)[1+level];
    alpha = gel(dom1, idx);
#ifdef MPI
```

```
    if (typ(alpha) != t_COMPLEX)
      alpha = gadd(alpha, mulcxI(real_0(prec)));
    memcpy(complex_buf, gel(alpha, 2),
          lg(gel(alpha, 2)) * sizeof(ulong));
    memcpy(complex_buf + lg(gel(alpha, 2)),
          gel(alpha, 1), lg(gel(alpha, 1)) * sizeof(ulong));
    MPI_Bcast(complex_buf, lg(gel(alpha, 1)) + lg(gel(alpha, 2)),
        MPI_LONG, peer_idx, MPI_COMM_WORLD);
  }
  else {
    alpha = alpha_buf;
    MPI_Bcast(complex_buf, 2*prec + 2, MPI_LONG,
               peer_idx, MPI_COMM_WORLD);
    gel(alpha, 1) = complex_buf + lg(complex_buf);
    idx = -1;
#endif
  }
#ifndef VIEUX
  aalpha = gnorm(alpha);
  lognalpha = glog(aalpha, prec);
  aalpha = gsub(aalpha, uno);
  calpha = gconj(alpha);
#else
  aalpha = gabs(alpha, prec);
  calpha = gdiv(aalpha, alpha);
  lognalpha = gdiv(alpha,
         gmul(gsqrt(aalpha, prec), gadd(uno, aalpha)));
  coeff3 = ginv(aalpha);
  coeff2 = gneg(gmul2n(gadd(aalpha, coeff3), -1));
  coeff3 = gneg(gmul2n(gsqr(gsub(aalpha, coeff3)), -2));
#endif

  dom2 = gel(dom, 2);
#ifdef VIEUX
#define FORMULA(z)                    \
  ({GEN divs=gmul((z),calpha);        \
   gmul(lognalpha, gsub(gadd(uno, divs),\
           gsqrt(gadd(gsqr(            \
                gadd(divs, coeff2)), coeff3), prec)));})
#else
#define FORMULA(z)      gsub(uno,      \
        gdiv(lognalpha, glog(          \
                gadd(uno, gdiv(aalpha,  \
                              gsub(uno, gmul(calpha, (z))))), prec)))
#endif

  zz = cgetg(lg1, t_VEC);
  (void)timer();
  for (i = 1; i < lg1; ++i)
  {
    if (i == idx) dom1++;
    gel(zz, i) = FORMULA(gel(dom1, i));
  }
  tpoints += timer();

  if (dom2 != gen_0) {
    long lfuncs = glength(dom2);

    f = cgetg(lg(dom2), t_VEC);
    precdl++;
    f1 = FORMULA(gel(dom2, 1));
    if (!valp(f1)) {
      gel(f1, 2) = gen_0;
      f1 = normalize(f1);
    }
    /*gel(f1, 2) = greal(gel(f1, 2));*/
    precdl--;
    gel(f, 1) = f1;

    for (i = 2; i <= lfuncs; i++) {
      gel(f, i) = FORMULA(gel(dom2, i));
    }
    tseries += timer();
    lbreak();
  }

  rdom = cgetg(4, t_VEC);
  gel(rdom, 1) = zz;
  gel(rdom, 2) = dom2 == gen_0 ? gen_0 : f;
  gel(rdom, 3) = gel(dom, 3);

  return rdom;
}

void
usage(char *name, FILE *f)
```

```
{
  fprintf(f, "Usage:␣%s␣[options]\n", name);
  fprintf(f, "Where␣options␣are:\n");
  fprintf(f, "␣␣-h␣␣␣␣␣␣␣␣␣␣:␣get␣a␣little␣help\n");
  fprintf(f, "␣␣-l␣<real>␣␣␣:␣lambda\n");
  fprintf(f, "␣␣-t␣<real>␣␣␣:␣t\n");
  fprintf(f, "␣␣-m␣<integer>:␣allocated␣memory\n");
  fprintf(f, "␣␣-M␣<integer>:␣log2(allocated␣memory)\n");
  fprintf(f, "␣␣-p␣<integer>:␣real␣precision\n");
  fprintf(f, "␣␣-s␣<integer>:␣series␣precision\n");
  fprintf(f, "␣␣-P␣<integer>:␣number␣of␣processes\n");
  fprintf(f, "␣␣-f␣<file>␣␣␣:␣restart␣from␣<file>\n");
  fprintf(f, "␣␣-c␣<integer>:␣step␣of␣the␣computation");
          fprintf(f, "␣(useful␣with␣-f)\n");
  fprintf(f, "␣␣-N␣␣␣␣␣␣␣␣␣␣:␣be␣nasty␣(nice␣0␣instead␣of");
          fprintf(f, "␣18)\n");
  fprintf(f, "␣␣-u␣␣␣␣␣␣␣␣␣␣:␣choose␣progressive␣succession");
          fprintf(f, "␣of␣alpha\n");
  fprintf(f, "␣␣-r␣␣␣␣␣␣␣␣␣␣:␣choose␣pseudo-random␣succession");
          fprintf(f, "␣of␣alpha\n");
}

int
main(int argc, char **argv)
{
    pari_sp ltop, limit;
    int i, optloop;
    char *lambda = "5.23";
    char *recup = NULL;
    char *dirname = 0;
    int prec = MEDDEFAULTPREC;
    unsigned long mem = 128 * 1024 * 1024;
    int dl = -1;
    unsigned long powof2 = 1;
    long log2ofpow = 0;
    int nice = 18, o;
    struct timeval tvstart, tvstop;
    long lg1;

    umask(022);
    gettimeofday(&tvstart, NULL);
    iter = 0;
    optloop = 1;
```

```
#ifdef MPI
      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &np);
      MPI_Comm_rank(MPI_COMM_WORLD, &level);
/*    printf("Pid %d, rank %d of %d\n", getpid(), level, np); */
      alpha_buf = (GEN) pari_malloc((2*(prec+1)+3)*sizeof(ulong));
      complex_buf = alpha_buf + 3;
      alpha_buf[0] = evaltyp(t_COMPLEX) | evallg(3);
      gel(alpha_buf, 2) = complex_buf;
#endif
      while (optloop) {
        switch(o = getopt(argc, argv, "hl:m:M:p:s:f:Nc:")) {
          case 'h':
              usage(argv[0], stdout);
              exit(0);

          case 'l':
              lambda = optarg;
              break;

          case 'm':
              { char *endptr;

              mem = strtol(optarg, &endptr, 0);
              if (endptr) {
                switch (*endptr) {
                  case 'k': case 'K':
                    mem <<= 10;
                    break;
                  case 'm': case 'M':
                    mem <<= 20;
                    break;
                  case 'g': case 'G':
                    mem <<= 30;
                    break;
                }
              }}
              break;

          case 'M':
              mem = 1 << strtol(optarg, NULL, 0);
              break;
        case 'p':
            prec = 3 + log(10) * strtol(optarg, NULL, 0) /
                                      log(2) / BITS_IN_LONG;
            break;

        case 's':
            dl = strtol(optarg, NULL, 0);
            break;

        case 'f':
            recup = optarg;
            break;

        case 'N':
            nice = 0;
            break;

        case 'c':
            iter = strtol(optarg, NULL, 0);
            break;

        case -1:
            optloop = 0;
            break;

        case '?':
        case ':':
            usage(argv[0], stderr);
            exit(1);
      }
    }
    setpriority(PRIO_PROCESS, 0, nice);
    pari_init(mem, 500000);
    (void)pol_x(fetch_user_var("y"));
    (void)pol_x(fetch_user_var("z"));
    (void)pol_x(fetch_user_var("t"));
    (void)pol_x(fetch_user_var("u"));
    (void)pol_x(fetch_user_var("q"));

    signal(SIGUSR1, (void*)&usr1handler);

#ifdef SIGHUP
    hdlr[SIGHUP] = signal(SIGHUP, (void*)&save_state);
```

```
#endif
#ifdef SIGINT
    hdlr[SIGINT] = signal(SIGINT, (void*)&save_state);
#endif
#ifdef SIGTERM
    hdlr[SIGTERM] = signal(SIGTERM, (void*)&save_state);
#endif
#ifdef SIGQUIT
    hdlr[SIGQUIT] = signal(SIGQUIT, (void*)&save_state);
#endif
#ifdef SIGIOT
    hdlr[SIGIOT] = signal(SIGIOT, (void*)&save_state);
#endif
#ifdef SIGBUS
    hdlr[SIGBUS] = signal(SIGBUS, (void*)&save_state);
#endif
#ifdef SIGFPE
    hdlr[SIGFPE] = signal(SIGFPE, (void*)&save_state);
#endif
#ifdef SIGXCPU
    hdlr[SIGXCPU] = signal(SIGXCPU, (void*)&save_state);
#endif
#ifdef SIGPWR
    hdlr[SIGPWR] = signal(SIGPWR, (void*)&save_state);
#endif
#ifdef SIGUSR2
    hdlr[SIGUSR2] = signal(SIGUSR2, (void*)&save_state);
#endif
    if (dl != -1) {
      precdl = dl;
    }
    ltop = avma;
    limit = stack_lim(avma, 2);
    precreal = prec;
    /*sleep(10);*/
    if (recup) {
      char buf[1000];
      char *p;

      cur_dom = NULL;
      if (np > 1 && strlen(recup) < sizeof(buf)-10) {
        if ((p = strchr(recup, '@'))) {
          *p = '\0';
      p += 2;
      while (isdigit(*p)) {
          p++;
      }
      sprintf(buf, ”%s@l%d%s”, recup, level, p);
      if ((p = strchr(buf, '/'))) {
        *p = '\0';
        dirname = strdup(buf);
        *p = '/';
      }
      recup = buf;
    }
  }
  else if ((p = strchr(recup, '/'))) {
    *p = '\0';
    dirname = strdup(recup);
    *p = '/';
  }
  if (!cur_dom)
      cur_dom = gp_read_file(recup);
  if (typ(gel(cur_dom, 1)) == t_INT) {
    iter = itos(gel(cur_dom, 1));
    cur_dom = gel(cur_dom, 2);
  }
  else {
    char *num;

    if (!iter && (num = strrchr(recup, '-'))) {
      log2ofpow = atoi(num);
      powof2 = 1 << -log2ofpow;
      iter = powof2 >> 1;
    }
  }
  gel(cur_dom, 3) = gclone(gel(cur_dom, 3));
  precdl = glength(gmael(cur_dom, 3, 4))+1;
  precreal = prec = lg(gmael(cur_dom, 3, 1));
  uno = gclone(real_1(prec));
}
else {
    uno = gclone(real_1(prec));
    cur_dom = gerepilecopy(ltop,
            domain(strtor(lambda, prec), prec));
}
```

```
/*printf("l%d: %d\n", level, glength(gel(cur_dom, 1))); */
/*printf("al%d %08x %08x %08x %08x\n", level,    \
 ((unsigned int *)gel(cur_dom, 1))[0],           \
 ((unsigned int *)gel(cur_dom, 1))[1],           \
 ((unsigned int *)gel(cur_dom, 1))[2],           \
 ((unsigned int *)gel(cur_dom, 1))[3]); */
/*printf("f%d\n", level); */
    if (dirname)
      chdir(dirname);
    else
    {
        char *str = GSTR(gmael(cur_dom, 3, 3)), *str2;
        char buf[1000];

        str2 = strchr(str, '-');
        if (str2) {
            *str2 = '\0';
            if (np > 1) {
              sprintf(buf, "%s@l%d", str, level);
              str = buf;
            }
            mkdir(str, 0777);
            chdir(str);
            *str2 = '-';
        }
    }
    {
        char buf[1024];

        /*sprintf(buf, "out-%d.", level);
        gethostname(buf+strlen(buf),
        sizeof(buf)-strlen(buf)-1);*/
        strcpy(buf, "out");
        (void)freopen(buf, "a", stdout);
        memcpy(buf, "err", 3);
        (void)freopen(buf, "a", stderr);
    }
    if (iter) {
      if (2 * iter != powof2) {
        powof2 = 1;
        log2ofpow = 0;
        while (powof2 <= iter) {
          powof2 <<= 1;
        log2ofpow--;
      }
    }
  }
  else {
    iter = 1;
  }
  if (iter == 1)
    writebin(GSTR(concat(gmael(cur_dom, 3, 3),
                    strtoGENstr("-0"))), cur_dom);
  lg1 = gmael(cur_dom, 3, 8)[lg(gmael(cur_dom, 3, 8))-1];
  for (; iter <= lg1; iter++) {
    cur_dom = stepkoebe(cur_dom, prec);
    if (iter == 50000) kill(getpid(), SIGUSR2);
    if (avma < limit) {
      GEN dom = cur_dom;

      /*
       * So that save_state does not try to save an invalid
       * cur_dom.
       */
      cur_dom = NULL;
      cur_dom = gerepilecopy(ltop, dom);
    }
    if (iter == powof2) {
      char *s;
      time_t t;

      writebin(GSTR(concat(gmael(cur_dom,3,3),
                    stoi(--log2ofpow))), cur_dom);
      time(&t);
      s = ctime(&t);
      if (s[strlen(s)-1] == '\n')
          s[strlen(s)-1] = '\0';
      printf("%s %ld\n", s, -log2ofpow);
      fflush(stdout);
      powof2 <<= 1;
    }
    else if (iter == lg1 - 1000)
    {
      char *s;
      time_t t;
```

```
        writebin(GSTR(concat(gmael(cur_dom,3,3),
                        stoi(-1000))), cur_dom);
        time(&t);
        s = ctime(&t);
        if (s[strlen(s)-1] == '\n')
            s[strlen(s)-1] = '\0';
        printf("%s %ld\n", s, -log2ofpow);
        fflush(stdout);
      }
    }
    {
      char *filename;

      cur_dom = gerepilecopy(ltop, cur_dom);
      filename = GSTR(concat(gmael(cur_dom, 3, 3),
                     strtoGENstr("-final")));
      writebin(filename, cur_dom);
    }
    gettimeofday(&tvstop, NULL);
    printf("Time: %ld, points: %ld, series: %ld\n",
           1000*(tvstop.tv_sec - tvstart.tv_sec) +
               (tvstop.tv_usec-tvstart.tv_usec)/1000,
           tpoints, tseries);

    return 0;
}

/*
:se makeprg=make\ debug
 vi:aw
*/
```