Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Cómputo eficiente de números absolutamente normales

Tesis presentada para optar al título de
Licenciado en Ciencias de la Computación

Martin Epszteyn

Directora: Dra. Verónica Becher
Codirector: Lic. Pablo Ariel Heiber
Buenos Aires, 2014

CÓMPUTO EFICIENTE DE NÚMEROS ABSOLUTAMENTE NORMALES

Un número real es simplemente normal en una base si todos los dígitos ocurren con la misma frecuencia en la expansión del número en dicha base. Un número es absolutamente normal si es simplemente normal en toda base entera mayor o igual que dos. Mostramos cómo computar números absolutamente normales de forma eficiente. Nuestra implementación computa un número absolutamente normal con una complejidad de tiempo parecida a $O(n^2)$, donde n es la cantidad de símbolos computados. Ésta es la primer implementación eficiente de un algoritmo que computa un número absolutamente normal. Exhibimos los primeros dos millones y medio de dígitos de la expansión binaria de cuatro números absolutamente normales que computamos. Medimos la discrepancia de normalidad en varias bases de esos resultados y de otros números conocidos. Mostramos también representaciones gráficas de las expansiones de los números analizados en diferentes bases.

Palabras claves: Números normales, Normalidad de Borel, Normalidad Absoluta, Distribución uniforme, cómputo eficiente

EFFICIENT COMPUTATION OF ABSOLUTELY NORMAL NUMBERS

A number is simply normal to a given base if every digit occurs with the same frequency in the expansion of the number in that base. A number is absolutely normal if it is simply normal to every integer base greater than or equal to two. We show how to compute absolutely normal numbers efficiently. Our implementation computes an absolutely normal number with time complexity similar to $O(n^2)$ with respect to the number of symbols produced. This is the first efficient implementation of an algorithm that computes an absolutely normal number. We exhibit the initial two and a half million digits of the binary expansion of four different absolutely normal numbers that we computed. We measure the discrepancy of normality to several bases of these results and of other known numbers. We also show graphical representations of the expansions of the analyzed numbers in different bases.

Keywords: Normal Numbers, Borel Normality, Absolute Normality, Uniform Distribution, efficient computation

Índice general

1.. Introducción 1
1.1. Principales resultados 2

2.. Preliminares 5

3.. Un algoritmo eficiente 7

4.. Implementación eficiente 9
4.1. Longitud de la extensión y cota superior de la discrepancia 10
4.2. El rol de la secuencia de entrada 10
4.3. Intervalos b-ádicos 11
4.4. Detalles de implementación 11
4.5. Cómputo 12
4.5.1. Secuencias de entrada y salida 12
4.5.2. Dígitos descartados de la secuencia de entrada 12
4.5.3. Tiempos de Cómputo 14

5.. Visualización 17

6.. Discrepancia Simple 19
6.1. Máxima discrepancia aceptable por nuestro algoritmo 19
6.2. Discrepancia de las secuencias de entrada y las de salida 21
6.3. Comparación de las discrepacias computadas 21

1. INTRODUCCIÓN

Hace unos cien años Émile Borel [10] definió una noción de aleatoriedad para los números reales que llamo *normalidad absoluta* y escribió:

> "Dans l'état actuel de la Science, la détermination effective d'un nombre absolument normal paraît un problème des plus difficiles: il serait intéressant, soit de le résoudre en construisant un nombre absolument normal, ou en montrant qu'un nombre irrationnel connu est absolument normal, soit de démontrer que, parmi les nombres pouvant être réellement définis, aucun n'est absolument normal; si paradoxal que paraisse cet énoncé, il n'est nullement incompatible avec le fait que la probabilité pour qu'un nombre soit absolument normal est égale à l'unité."

Llamamos *base* a un entero mayor o igual que 2. Un número real x cuya expansión in base b está dada por

$$x = \lfloor x \rfloor + \sum_{j \geqslant 1} a_j \, b^{-j}, \text{ donde } a_j \in \{0, 1, \ldots, b-1\},$$

es *simplemente normal* en base b si cada dígito $d \in \{0, 1, \ldots, b-1\}$ aparece en la secuencia$(a_j)_{j \geqslant 1}$ con la misma frecuencia $1/b$. Es decir, por cada uno de esos dígitos d,

$$\lim_{n \to \infty} \frac{\#\{j : 1 \leqslant j \leqslant n \text{ y } a_j = d\}}{n} = \frac{1}{b}.$$

Un número x es *normal* en base b si es simplemente normal en base b^k para cada entero $k \geqslant 1$. Un número es *absolutamente normal* si es normal en todas las bases. Esta definición no es la que Borel dio originalmente, pero es equivalente a esta. Una buena exposición de varias formulaciones equivalentes se puede encontrar en [11].

Exhibimos aquí, por primera vez desde que Borel presentó el problema, el cómputo de instancias particulares de números absolutamente normales. Damos una implementación eficiente de un algoritmo polinomial reciente dado por Becher, Slaman y Heiber [7]. Dicho algoritmo computa un número absolutamente normal en tiempo apenas superior a cuadrático.

La existencia de números absolutamente normales fue demostrada por Borel también en [10] mostrando que casi todos los reales en el intervalo unitario son absolutamente normales. En 1917 el *Bulletin de la Societé Mathématique de France* publicó en el mismo volumen dos construcciones, una de Henri Lebesgue y otra de Wacław Sierpiński. Sin embargo, ninguna de las dos estaba basada en cálculos finitarios, y ninguno de los dos números definidos podían ser exhibidos. La primer construcción computable corresponde a Alan Turing [23, 5]. Ni el algoritmo de Turing ni la reformulación computable de la construcción de Sierpiński [4] sirven para exponer un ejemplo de un número absolutamente normal puesto que ambas tiene complejidad doblemente exponencial. Hay también una construcción computable de Wolfgang Schmidt [21] basada en análisis armónico, y una de Mordechay Levin [15], pero la complejidad computacional de esas construcciones no fue analizada. Aunque se sabía de la existencia de números absolutamente normales computables, hasta hace poco no había algoritmos eficientes en teoría ni utilizables en la práctica.

El problema de exhibir un número normal en una base dada fue resuelto con éxito por David Champernowne[1] En 1933 [12], el mostró que el número real

$$0{,}12345678910111213\ldots$$

obtenido concatenando la representación decimal de todos los enteros positivos en orden es normal en base 10. No se sabe aun si es o no normal en base que no sean potencias de 10. El método de construcción de número normales en una base por concatenación de dígitos se ha vuelto estándar. Un nuevo desarrollo concatenando dígitos de funciones aritméticas fue hecho por Paul Pollack y Joseph Vandehey [19].

Maxfield en 1953 [17] demostró que un número real es normal en una base exactamente cuando es normal en toda base *multiplicativamente dependiente* de esa base. Recordemos que dos enteros positivos s y t son *multiplicativamente dependientes* cuando existen enteros positivos n y m tales que s^n es igual a t^m. Por ejemplo, 4 y 8 son multiplicativamente dependientes, pero 4 y 6 no lo son. Schmidt [21] mostró que esta es la única restricción sobre las bases en las cuales un número real puede ser normal. El demostró que para cualquier conjunto de bases, cerrado bajo dependencia multiplicativa, existen números normales en toda base presente en el conjunto y simultáneamente no normales en toda base en el complemento del mismo. Schmidt también dio una construcción explícita para proveer instancias. Recientemente Bailey y Crandall [2] mostraron que las constantes de Stoneham, que son los reales de forma

$$\alpha_{b,c} = \sum_{n \geqslant 1} 1/(c^n b^{c^n})$$

para enteros coprimos b y c ambos mayores o iguales que 2, son normales en base b pero no normales en base bc.

En 1950, Borel [9] conjeturó que todo número algebraico irracional (cada raíz irracional de un polinomio algebraico con coeficientes enteros) es normal en todas las bases enteras positivas. El trabajo reciente [18] hace investigaciones empíricas en la distribución de los dígitos de muchos números algebraicos, incluyendo lo números de Pisot-Vijayaraghavan, lo números de Salem y los números algebraicos que ocurren como las mayores raíces reales de ciertos polinomios con coeficientes elegidos al azar. Se cree también que otras constantes matemáticas usuales como e y π son absolutamente normales. El experimento realizado por Davis Bailey y Richard Crandall [3], y más recientemente Francisco Aragon Artacho, David Bailey y Jonathan y Peter Borwein [1] provee evidencia empírica de que estas constantes presentan baja discrepancia en segmentos iniciales largos, es decir, que lucen absolutamente normales hasta el punto donde se pudo observar.

1.1. Principales resultados

Mostramos como computar segmentos iniciales de longitud arbitraria de la expansión binaria de números absolutamente normales. Presentamos una implementación del algoritmo de Becher, Slaman y Heiber [7], con una secuencia binaria como parámetro adicional. Exhibimos el cómputo para cuatro diferentes valores de dicho parámetro:

- la expansión en base 2 de π,

[1] Champernowne trabajó bajo la supervisión de H.G. Hardy in el King's College, Cambridge, UK.

- la expansión en base 2 del número de Champernowne,
- la expansión en base 2 del número de Stoneham $\alpha_{2,3}$,
- la expansión en base 2 de un número pseudoaleatorio.

La Sección 3 presenta la implementación del algoritmo. El código fuente del programa está disponible en `http://www.dc.uba.ar/people/profesores/becher/software/ann.zip`

Por cada uno de estos parámetros de entrada hemos computado 2500000 dígitos en binario de la salida. Almacenamos datos estadísticos de la discrepancia de las secuencias de salida obtenidas con los cuatro valores del parámetro. Los comparamos con los datos estadísticos de la discrepancia de las mismas secuencias de entrada. En la Sección 6 presentamos estos datos.

Podemos observar que la complejidad temporal del algoritmo parece ser cuadrática. En la Sección 4.5.3 presentamos estos datos.

Verificamos en la sección 6 que la discrepancia de la salida cumple con las cotas esperadas. Además vemos datos empíricos consistentes con la demostración de Becher y Slaman [8] de que las funciones de discrepancia para las bases multiplicativamente independientes en las cuales un número es normal son independientes de a pares.

2. PRELIMINARES

Notación. Llamamos *base* a un entero mayor o igual que 2. Dado un número real x, escribimos $(x)_b$ para indicar su expansión en base b. Usamos al frase *intervalo b-ádico* para referirnos a un intervalo semiabierto I de forma $[a/b^m, (a+1)/b^m)$, para enteros a y m que cumplen $0 \leqslant a < b^m$.

Un *dígito* en base b es un elemento de $\{0, ..., b-1\}$ y un *bloque* en base b es una secuencia finita de dígitos en base b. Dado un bloque u, $|u|$ es su longitud, $u[i]$ es el dígito en la posición i y $u[i..i+n-1]$ es el bloque interno de n dígitos consecutivos en el bloque u empezando desde la posición i, donde $1 \leqslant i \leqslant |u|-n+1$. Pasamos libremente entre intervalos b-ádicos y bloques en base b que representan racionales. Si u es un bloque en base b y se entiende que estamos trabajando en base b, entonces $.u$ denota al número racional cuya expansión en base b tiene exactamente los dígitos que aparecen en u. Dado el bloque u, los reales con representación en base b cuya secuencia de dígitos extienden u son exactamente aquellos que pertenecen al intervalo b-ádico $[.u,\ .u+b^{-|u|})$. Además, cada intervalo b-ádico $[a/b^m, (a+1)/b^m)$ corresponde a un bloque u como antes, dónde u se obtiene de escribir a en base b y luego anteponerle suficientes ceros para obtener un bloque de longitud m.

En lo que sigue utilizamos la notación O grande estándar de ciencias de la computación, que muestra el comportamiento asintótico de una función dada. Una función g es $O(f(n))$ cuando existen constantes n_0 y c tales que para todo $n \geqslant n_0$, $g(n) \leqslant cf(n)$. Escribimos log para referirnos al logaritmo en base 2.

Un dígito d *ocurre* en un bloque u en la posición i si $u[i] = d$. Dados un bloque u y un dígito d, definimos el número de ocurrencias de d en u $occ(u, d) = \#\{i : u[i] = d\}$.

La *discrepancia simple* de un segmento inicial en la representación en base b de un número real indica la cantidad por la cual los números del segmento inicial varían de su promedio esperado. Presentamos la definición para un bloque de dígitos y una base.

Definition. La *discrepancia simple* de un bloque u en base b es

$$D(u,b) = \text{máx}\left\{\left|\frac{occ(u,d)}{|u|} - \frac{1}{b}\right| : d \in \{0,1,\ldots,b-1\}\right\}.$$

Nótese que $D(u, b)$ es un número entre 0 y $1 - 1/b$. Una presentación de discrepancia para normalidad puede encontrarse en [14, 11].

La definición de normalidad se puede expresar usando discrepancia: un número real x es simplemente normal en base b si $\lim_{n\to\infty} D((x)_b[1..n], b) = 0$. Como la normalidad absoluta es normalidad simple en todas las bases, tenemos lo siguiente.

Lemma. *Un real x es absolutamente normal si, y sólo si, para toda base b,*

$$\lim_{n\to\infty} D((x)_b[1..n], b) = 0.$$

3. UN ALGORITMO EFICIENTE

El algoritmo de Becher, Slaman y Heiber [7] se basa en la caracterización de normalidad absoluta usando discrepancia, dada en el Lema 2, y tiene complejidad apenas arriba de cuadrática.

Theorem ([7], Teorema 4.3)**.** *Sea f una función computable no decreciente no acotada. Existe un algoritmo que computa un número absolutamente normal x tal que, por cada base b, devuelve los primeros n dígitos de la representación en base b de x usando $O(f(n)\ n^2)$ operaciones elementales.*

El teorema afirma que es posible conseguir velocidad de cómputo. Observando el trabajo, puede verse que dicha velocidad de cómputo se obtiene a cambio de sacrificar la velocidad en la convergencia a normalidad, haciendo muy lentamente tanto la incorporación de nuevas bases como la reducción de discrepancia.

La salida del algoritmo dado en [7] es una secuencia de ceros y unos que representa la expansión binaria de un número absolutamente normal en el intervalo unitario. En cada paso el algoritmo extiende la expansión actual con una nueva secuencia de ceros y unos recientemente calculada. En el límite computa una expansión infinita.

Supongamos que el bloque u representa los primeros n dígitos binarios del número real x. Entonces el real x representado por la expansión infinita obtenida como límite del cómputo, pertenece al intervalo diádico $[.u, .u + 2^{-|u|})$. De esta manera, el cómputo realizado por el algoritmo es un refinamiento iterativo de intervalos diádicos. Cada paso determina un nuevo subintervalo. El resultado x es el único real contenido en la intersección de todos los subintervalos calculados.

La salida luego de i pasos es un bloque de dígitos binarios, obtenido como la concatenación de extensiones encontradas en todos los pasos anteriores, y este bloque es la expansión binaria del extremo izquierdo del intervalo diádico calculado en el paso i. Luego, el bloque de dígitos resulta ser un prefijo en la expansión de x.

En cada paso el algoritmo controla que una cota máxima para la discrepancia simple en una cantidad finita de bases. Dicha cota se reduce hacia un límite de 0 en los sucesivos pasos del algoritmo. La correctitud del algoritmo viene de que siempre existe una extensión del bloque actual con baja discrepancia en cada una de las bases consideradas. Para cumplir eso, la extensión debe ser lo suficientemente larga.

El algoritmo establece como funciones del número de paso i, la cantidad de bases revisar, la cantidad de dígitos que debe tener la extensión, y la cota máxima de discrepancia aceptable en la extensión. Estas se llaman en [7], t_i, ε_i y k_i, respectivamente. En el paso i, el algoritmo elige un bloque binario de longitud k_i cumpliendo para cada base $b \leqslant t_i$, que los $\lceil k_i / \log b \rceil$ dígitos en base b que son elegidos en este paso contengan cada dígito individual a lo sumo $k_i/b + \varepsilon_i$ veces.

En [7] se demuestra que existen elecciones para las funciones t_i, ε_i y k_i que aseguran tanto la normalidad absoluta de la salida como la complejidad de tiempo del algoritmo. Para conseguir velocidad computacional, la consideración de nuevas bases puede ser postergada, y el tope de la discrepancia puede bajar muy lentamente.

La normalidad absoluta requiere que t_i converja a infinito, ε_i converja a cero y k_i sea lo suficientemente grande como para asegurar que muchos bloques de longitud k_i tengan

discrepancia a lo sumo ε_i vistas como extensiones en bases hasta t_i. Además, k_i no debe ser demasiado grande, para evitar fluctuaciones de la discrepancia dentro de la extensión. En el paso $i+1$, una fracción grande, $1-\delta_{i+1}$, de todos los bloques de longitud k_{i+1} son extensiones aceptables donde

$$\delta_{i+1} = \frac{1}{8\ t_i}\ \frac{1}{2^{t_i+t_{i+1}} t_i! t_{i+1}!},$$

y

$$\text{máx}\left(\left\lceil \frac{6}{\varepsilon_{i+1}} \right\rceil, \left\lceil -\ln\left(\frac{\delta_{i+1}}{2t_i}\frac{6}{\varepsilon_{i+1}^2}\right)\right\rceil\right) < k_{i+1} < \varepsilon_{i+1}|\log(|I_i|)|.$$

El algoritmo en [7] fija k_{i+1} en el mínimo de esta desigualdad y en cada pasao elige el intervalo viable de más a la izquierda.

4. IMPLEMENTACIÓN EFICIENTE

Damos aquí una implementación práctica del algoritmo de Becher, Slaman y Heiber [7]. Con el objetivo de aumentar la velocidad de cómputo, diferimos de las decisiones tomadas en [7] en dos puntos. En primer lugar, en vez de usar el límite teórico establecido para k_{i+1}, buscamos la primer longitud para la cual encontremos una extensión aceptable empezando desde la longitud utilizada en el paso anterior. Segundo, utilizamos un parámetro de entrada como oráculo para elegir una extensión aceptable. Además fijamos la máxima base t_i a ser considerada en el paso i como una función logarítmica en i, y la cota ε_i de discrepancia es la inversa de t_i. De esta manera, la incorporación de bases y la reducción de discrepancia son demoradas lo suficiente como para encontrar una cantidad significativa de dígitos en tiempo razonable, pero no tanto como para que los dígitos obtenidos consideren un subconjunto demasiado chico de bases o una cota de discrepancia demasiado alta como para resultar representativos de la absoluta normalidad. El Algoritmo 1 presenta el pseudocódigo de la implementación.

Algorithm 1 Algoritmo que recibe como parámetro una secuencia binaria

Paso de inicialización 0. Sea $I_0 = [0, 1)$ y $k_0 = 1$.

Paso recursivo $i + 1$. Dados t_i, k_i, I_i y $u_{i,b}$ para $b = 2, \ldots, t_i$,

1. Sea $t_{i+1} = \lfloor 3 \log(i + 2) \rfloor + 3$ y $\varepsilon_{i+1} = 1/t_{i+1}$.
2. Empezando con $k_{i+1} = k_i$.
3. Partir el intervalo I_i en $2^{k_{i+1}}$ subintervalos diádicos de igual longitud.
4. Sea $J_{i+1,2}$ uno de esos subintervalos elegido de acuerdo a los próximos dígitos la secuencia de entrada Sea $u_{i+1,2}$ el bloque de dígitos en base 2 tal que

$$J_{i+1,2} = [.u_{i+1,2},\ .u_{i+1,2} + 2^{-|u_{i+1,2}|}).$$

5. Por cada base $b = 3, \ldots, t_{i+1}$, sea $J_{i+1,b}$ un subintervalo b-ádico de $J_{i+1,b-1}$ de tamaño máximo. Sea $u_{i+1,b}$ el bloque en base b tal que

$$J_{i+1,b} = [.u_{i+1,b},\ .u_{i+1,b} + b^{-|u_{i+1,b}|}).$$

6. Por cada base $b = 2, \ldots, t_i$, sea $v_{i+1,b}$ tal que $u_{i,b}v_{i+1,b} = u_{i+1,b}$.
7. Si la discrepancia simple de algún $v_{i+1,b}$ es mayor que ε_{i+1}, volver al paso 3, incrementando k_{i+1} en 1 sólo en caso que $k_{i+1} < \varepsilon_{i+1}|u_{i,2}|$ o en caso que $\varepsilon_{i+1}\ i < 2$.
8. En caso contrario (las discrepancias simples de todos los $v_{i+1,b}$ son a lo sumo ε_{i+1}), sea I_{i+1} un subintervalo diádico de $J_{i+1,t_{i+1}}$ de tamaño máximo.

4.1. Longitud de la extensión y cota superior de la discrepancia

En cada paso el Algoritmo 1 controla que la discrepancia simple se mantenga por debajo de una cota máxima. Dicha cota máxima decrece de una manera fija para una cantidad finita de bases. El único requisito de la función ε_i es que converja a cero. Experimentamos asignar para ε_i los valores t_i^{-1} y t_i^{-2}. Con $\varepsilon_i = t_i^{-2}$ el cómputo era muy lento porque se requerían extensiones más largas para cumplir con la cota mas exigente de discrepancia. Por lo tanto elegimos utilizar $\varepsilon_i = t_i^{-1}$ que empíricamente dio un comportamiento más rápido.

La correctitud del algoritmo, demostrada en [7], se sigue de que siempre existe una extensión de la salida actual con baja discrepancia en cada una de las bases consideradas. La longitud de la extensión en el paso $i + 1$ es k_{i+1}, y no puede ser demasiado larga, siendo su máximo valor aceptable $\varepsilon_{i+1}|\log(|I_i|)|$. La longitud de la extensión tiene que ser suficientemente larga como para asegurar que haya una extensión aceptable. A medida que aumenta la longitud, la proporción de extensiones válidas aumenta exponencialmente. Hay una cota inferior teórica para k_{i+1} que seguro funciona, pero dicha cota es muy holgada, lo que implica que en la práctica se pueden encontrar extensiones de longitud mucho más corta. En cada paso i el número de operaciones elementales es una función de k_{i+1}, así que cuanto menor es k_{i+1} más rápido corre el algoritmo. Luego, menores longitudes ofrecen mejor performance. El Algoritmo 1 intenta primero con la longitud usada en el paso anterior y si no encuentra extensiones aceptables de esa longitud, la incrementa en uno. Si consideramos los dígitos de la entrada como algo suficientemente pseudoaleatorio, puede verse que, dado el aumento exponencial en la cantidad de extensiones aceptables, una vez alcanzada una longitud en la que existen dichas extensiones, la esperanza de la cantidad de tiempo en encontrar una de ellas eligiendo "al azar" no es demasiado grande.

4.2. El rol de la secuencia de entrada

Como solamente nos interesa obtener un número absolutamente normal, cualquiera que lo sea, la elección en cada paso de una extensión aceptable puede ser hecha de cualquier forma. En vez de buscar el bloque binario lexicográficamente menor que sea aceptable, como se hace en [7], podemos determinarlo con la secuencia de entrada, a modo de un oráculo o de una fuente de dígitos pseudoaleatorios. En el paso i una extensión es aceptable si satisface el límite de discrepancia para toda las bases menores o iguales que t_i. Una secuencia de entrada de dígitos binarios aleatorios sería una buena candidata. Como se mencionó previamente, suponiendo que la entrada es aleatoria, incrementando la longitud de las extensiones consideradas, aumenta la probabilidad de elegir una extensión aceptable.

El Algoritmo 1 incrementa la longitud de la extensión en uno y considera el próximo bloque de esa longitud en la secuencia de entrada. Si la entrada no es suficientemente aleatoria, existe el riesgo de que ninguno de los bloques tomados de la entrada sea aceptable, y que la longitud de la extensión alcance el máximo para ese paso. Un ejemplo de una de estas fuentes mal comportadas sería una secuencia de todos ceros. Al siempre elegir bloques de todos ceros, la discrepancia calculada será máxima y no estará por debajo de la cota especificada. Si esto sucede el algoritmo no avanzará.

Una forma de evitar esta falta de avance sería sofisticar el Algoritmo 1 de esta forma. En el paso $i + 1$, si la longitud k_{i+1} alcanzó su límite superior $\varepsilon_{i+1}|\log(|I_i|)|$, cambiamos el valor de k_{i+1} por el de su límite teórico inferior y hacemos una búsqueda por fuerza bruta entre los posibles bloques de esta longitud hasta encontrar una extensión aceptable. Por la asignación

$t_{i+1} = \lfloor 3\log(i+2) \rfloor + 3$ y $\varepsilon_{i+1} = 1/t_{i+1}$, el límite teórico para k_{i+1} es

$$k_{i+1} = \text{máx}\left(6t_{i+1}, \left\lceil -\ln\left(\frac{6t_{i+1}}{16\ t_i^2\ 2^{t_i+t_{i+1}}(t_i!)^2}\right)\right\rceil\right) + 1.$$

No incluimos esta posibilidad en la implementación que hicimos porque en los experimentos que realizamos siempre conseguimos avance, excepto para secuencias obviamente poco aleatorias como la secuencia de entrada de todos ceros y otras secuencias periódicas. En particular experimentamos con las constantes matemáticas π, e y $\sqrt{2}$, el número de Stoneham $\alpha_{2,3}$, muchas secuencias del estilo de Champernowne, y muchas secuencias pseudoaleatorias. Está fuera del alcance de este trabajo investigar cuál es el requerimiento de aleatoriedad sobre la entrada para asegurar que el algoritmo siempre avance. Es fácil ver que si la entrada es un número absolutamente normal seguro será así, aunque parece factible que el requerimiento sea aún mas débil.

4.3. Intervalos b-ádicos

Representamos un intervalo b-ádico como una 3-upla de enteros, $\langle a, b, m\rangle$ tal que b es una base, m es no negativo y $0 \leqslant a < b^m$. El intervalo representado es $[ab^{-m}, (a+1)b^{-m})$, el conjunto de todos los números reales cuyos primeros m dígitos en base b, vistos como un entero en base b, representan a.

El Algoritmo 1 requiere computar un subintervalo b-ádico de tamaño máximo de un intervalo s-ádico dado, para bases b y s. La existencia de este subintervalo con una cota inferior en su tamaño está demostrada en el Lemma 3.4 en [7], y la complejidad computacional de esta tarea en términos del número de operaciones elementales necesarias está demostrada en el Lemma 4.1 del mismo trabajo. Nuestra implementación del Algoritmo 1 simplifica este último lema, usando un número constante de operaciones aritméticas en los enteros de la tupla $\langle a, b, m\rangle$ que representa al intervalo. Esta simplificación empeora levemente la cota de complejidad temporal teórica de la operación porque usa operaciones costosas como multiplicación sobre enteros no acotados, pero por otro lado permite una representación mas concisa y menor cantidad total de operaciones, lo cual aprovecha mejor las optimizaciones de la biblioteca que utilizamos para representar enteros no acotados.

4.4. Detalles de implementación

Nuestra implementación del Algoritmo 1 está programada en lenguaje C utilizando la biblioteca GMP [1] para representar números enteros no acotados. Todos los programas fueron compilados usando GCC 4.6.3.

Todas las corridas fueros realizados con una máquina virtual bajo Ubuntu 12.04 con 4GB de RAM y un CPU AMD x4 955 BE.

El Algoritmo 1 escala para correr indefinidamente. Su complejidad temporal teórica es apenas por encima de cuadrática y usa una cantidad lineal de memoria. Desgraciadamente nuestra implementación no puede alcanzar bases arbitrariamente grandes. Esta limitación viene de la biblioteca GMP que maneja representaciones hasta bases menores a 63. Por lo tanto, nuestro programa solamente revisa la discrepancia hasta bases menores de 63, pero no más. Utilizando tecnología actual, es poco probable que nuestro programa pudiera correr

[1] `sudo apt-get install libgmp3-dev` para instalarla en Ubuntu

hasta alcanzar este límite: la memoria requerida supera la memoria de una computadora actual mucho antes de llegar a incluir la base 63.

Nuestro programa solamente utiliza enteros no acotados de la biblioteca GMP para representar a en las tuplas $\langle a, b, m\rangle$ que representan los intervalos. Las bases, longitudes de las extensiones y los valores de b y m en las tuplas $\langle a, b, m\rangle$ son enteros acotados de 64 bits. Los valores de bases estan acotados por 63, así que esto no representa una restricción adicional. Los valores relacionados con longitudes (como m) crecen linealmente en el tiempo, con lo cual una cota de 2^{63} no es una limitación para las escalas de tiempo que manejamos. Es decir, en las corridas que realizamos, no hay riesgo de incurrir en overflow de estas variables, aunque una teórica corrida infinita requeriría todas variables no acotadas.

4.5. Cómputo

4.5.1. Secuencias de entrada y salida

Para realizar los experimentos obtuvimos las secuencias de entrada de la siguiente manera.

- Usamos la secuencia de los dígitos π en base 10 disponible en

  `http://zenwerx.com/projects/pi-digits/pi/`

- Computamos la secuencia de Champernowne en base 10 con un script (disponible junto con el código fuente del programa principal).

- Computamos una secuencia de dígitos pseudoaleatorios dinámicamente con la función `mpz_urandom()` de la biblioteca GMP, y fijamos arbitrariamente la semilla 42.
  (ver `https://gmplib.org/manual/Integer-Random-Numbers.html`).

- Obtuvimos la secuencia de dígitos del número de Stoneham $\alpha_{2,3}$ en base 4 directamente de David Bailey (comunicación personal).

Una vez leída la entrada en base 10, es convertida por el programa a base 2. A partir de entonces se considera la entrada como una secuencia binaria.

El cambio de base se realiza con funciones de la biblioteca GMP, lo cual permite trabajar con entradas arbitrariamentre grandes.

En todos nuestros experimentos establecimos el fin del cómputo luego de obtener 2500000 dígitos binarios. La mayor base considerada para todas las entradas fue 25.

4.5.2. Dígitos descartados de la secuencia de entrada

El Algoritmo 1 ve a la secuencia de entrada como un stream de dígitos binarios del cual, en el paso i, bloques de longitud k_i pueden ser aceptados o descartados de acuerdo al efecto que tengan en la discrepancia de las extensiones finales del intervalo en las bases que estamos revisando en el momento. En cada paso se acepta exactamente un bloque, y pueden descartarse 0 o más bloques.

La Tabla 4.1 muestra la cantidad de dígitos descartados de las distintas secuencias de entrada en las corridas realizadas, hasta llegar a obtener 2500000 dígitos binarios en la salida.

En los pasos más tempranos del Algoritmo 1 , k_i tiene un valor pequeño y por lo tanto es fácil que los bloques de la entrada de longitud k_i sean descartados. A medida que k_i aumenta, disminuye la proporción de dígitos descartados de la entrada respecto de la cantidad de dígitos

Paso i	# bits de salida	# bits descartados de la fuente
	x(Champernowne)	
1	27	0
21	2056	3086
100	33622	29675
1000	1129444	317282
1250	1560416	257419
1500	2027921	355111
1739	2501582	387604
	x(π)	
1	27	0
21	2178	2971
100	31138	27721
1000	1143128	269944
1250	1575078	306624
1500	2043171	377415
1724	2501466	420829
	x(random)	
1	27	0
21	1937	2578
100	31834	29740
1000	1134429	257424
1250	1556115	298455
1500	2010971	340643
1748	2500425	391794
	x($\alpha_{2,3}$)	
1	27	0
21	2124	3815
100	33342	24324
1000	1121148	243619
1250	1540072	288500
1500	1988293	331816
1762	2500100	388131

Tabla 4.1: Bits descartados por paso, en las cuatro fuentes consideradas

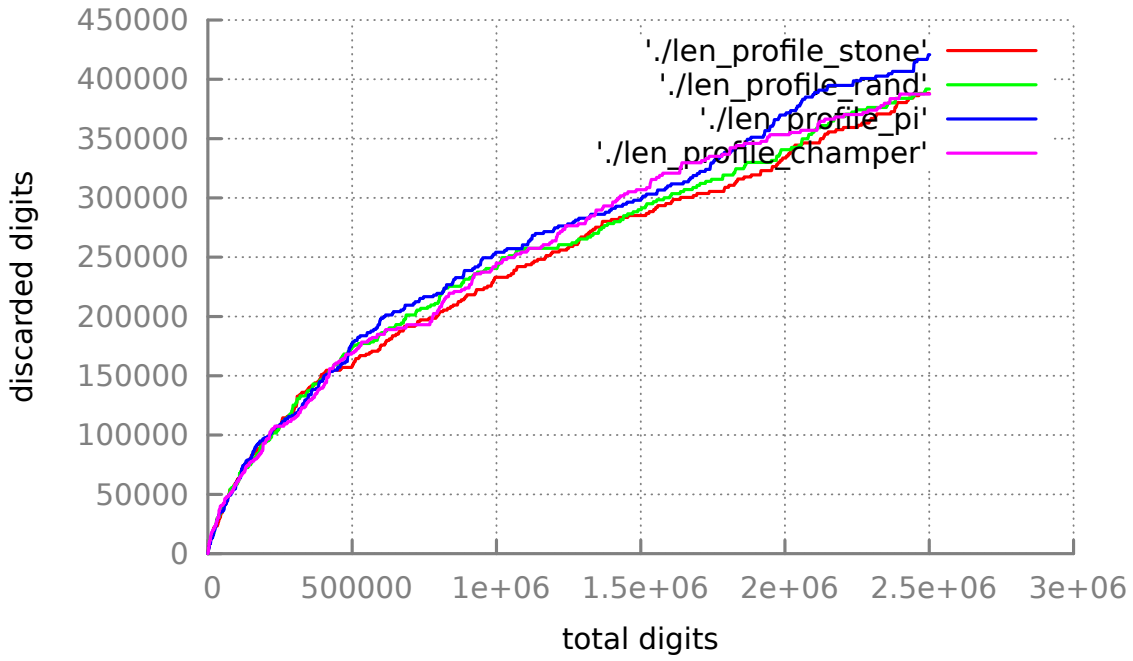

Figura 4.1: Número de dígitos descartados según longitud de la salida

en la salida, con lo cual el crecimiento de la cantidad de dígitos descartados es sublineal. Esto se corresponde con el crecimiento sublineal de las cotas teóricas para las longitudes k_i que garantizan la existencia de bloques aceptables.

4.5.3. Tiempos de Cómputo

La figura 4.2 muestra el tiempo de cómputo del algoritmo usando como entrada la secuencia de los dígitos de π. Con la invocación `./normal ‘‘pi.txt’’`. Vemos que el tiempo total de corrida fue de 12130 segundos (tres horas y casi veintitrés minutos) para producir 2 500 000 dígitos binarios. Con las otras secuencias de entrada obtuvimos tiempos de cómputo similares.

El estudio del algoritmo nos llevó a pensar en un modelo de complejidad temporal cuadrática en función de la cantidad de dígitos de salida. Obtener la complejidad exacta del algoritmo que utilizamos requiere un estudio mucho mas detallado del funcionamiento de la biblioteca GMP, y siendo un estudio de algorítmica teórica, queda fuera del alcance de este trabajo experimental.

Estimaciones de posibles funciones para los valores obtenidos con el programa Eureqa (`http://www.nutonian.com/index.php`) dieron resultados consistentes con este análisis, sugiriendo que la cantidad de segundos que toma el algoritmo para calcular n dígitos es similar a $1{,}93 \times 10^{-9} \times n^2$, con coeficiente de determinación de 0,99984226.

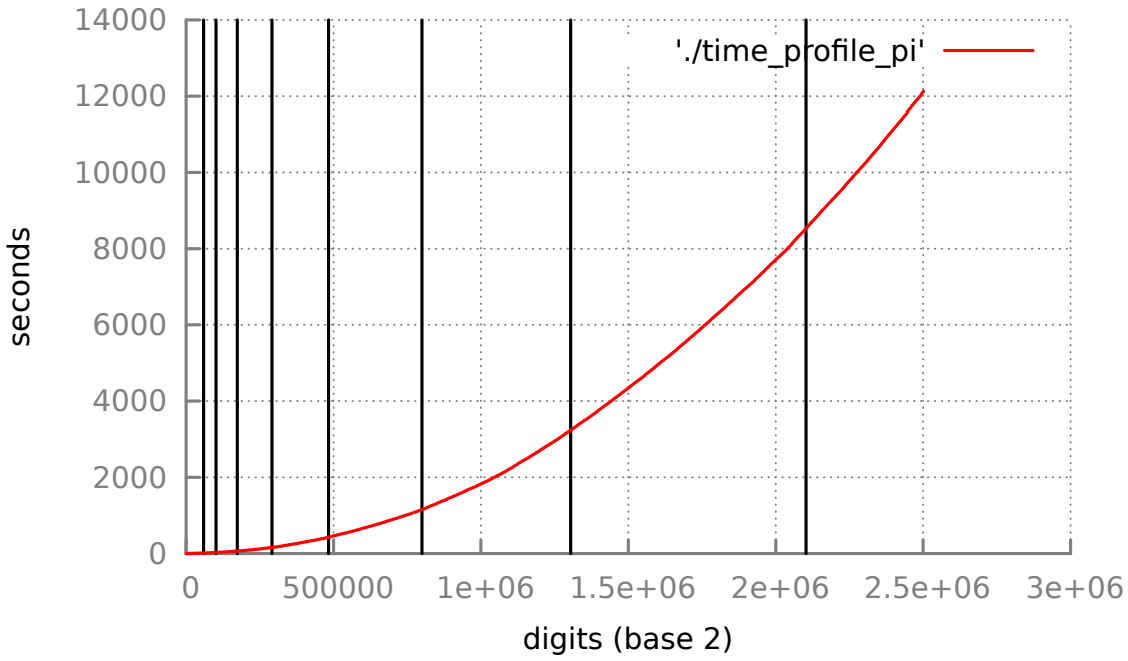

Figura 4.2: Tiempo para la generación de 2 500 000 dígitos binarios; las líneas verticales indican el momento a partir del cual se consideran las bases 18, 19, 20, 21, 22, 23, 24 y 25

5. VISUALIZACIÓN

Graficamos representaciones horizontales de los primeros 250000 dígitos de las cuatro secuencias de entrada y las cuatro secuencias de salida obtenidas con nuestros cómputos, en las bases 2, 6 y 10. La figura 5.1 muestra los 24 casos. Llamamos x(π), x(Champernowne), x(random) y x($\alpha_{2,3}$) a la salida de nuestro algoritmo tomando como entrada π, el número de Champernowne, un número pseudoaleatorio y $\alpha_{2,3}$ como entrada, respectivamente.

Para visualizar los dígitos de la expansión de un número en base b usamos una matriz de 500×500 píxeles, donde el píxel en la i-ésima fila y j-ésima columna representa el dígito $500i + j$ de la expansión en base b utilizando un color distinto para cada dígito. El que dos píxeles se encuentren uno a la derecha del otro significa que son consecutivos en la secuencia representada. Dos píxeles uno arriba del otro representan dos símbolos separados por otros 499 en el medio.

Un patrón visible en el dibujo es indicativo de un patrón en la secuencia representada. Cabe recordar que un número puede presentar patrones y ser absolutamente normal.

En base 6 podemos observar los bloques de ceros consecutivos de $\alpha_{2,3}$, representadas por las franjas de color rojo. En base 10 podemos observar el patrón de los dígitos del número de Champernowne, que tiene un comportamiento determinístico muy particular, a pesar de ser normal en esa base.

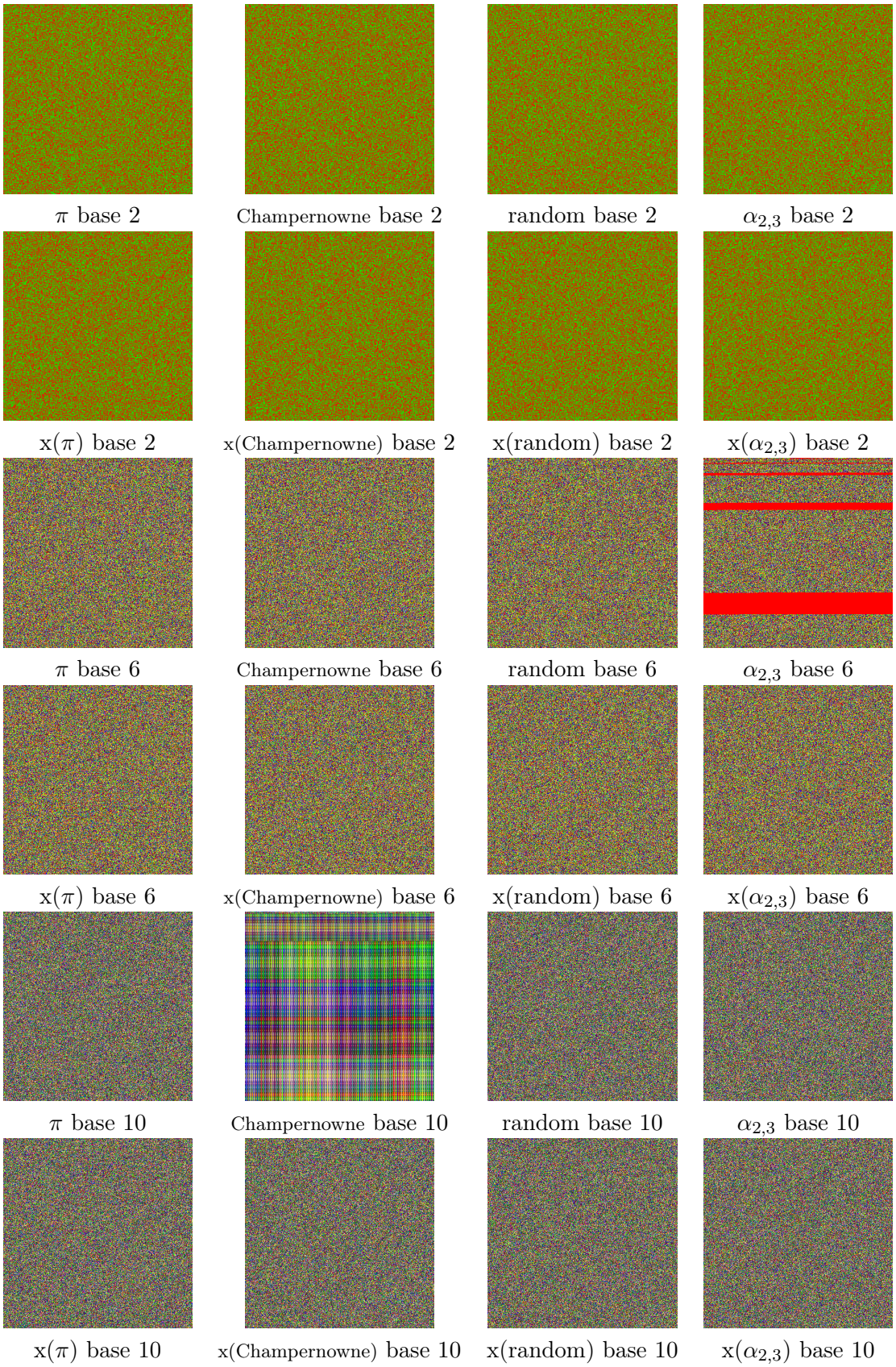

Figura 5.1: Representaciones horizontales de color

6. DISCREPANCIA SIMPLE

Los resultados conocidos sobre discrepancia analítica dan una cota superior para la discrepancia simple que consideramos aquí. Para una presentación del tema de discrepancia analítica ver las monografías [14, 11]. Los resultados demostrados por Gal y Gal [13] implican que casi todos los reales x en el intervalo unitario, cumplen para cada base b,

$$D((x)_b[1..n], b) = O\left(\sqrt{\frac{\log\log n}{n}}\right).$$

Levin [16] demostró que existe un número real x tal que

$$D((x)_b[1..n], b) = O\left(\frac{\log^2 n}{n}\right).$$

La discrepancia analítica del número de Champernowne y las de otros números parecidos al de Champernowne (como los de la familia definida por Davenport y Erdős [**?**]) fueron demostradas por Schiffer [20]. Se demostró que si x es de esos números, entonces se cumple que

$$D((x)_b[1..n], b) = O\left(\frac{1}{\log n}\right).$$

Schiffer de hecho demostró que esa cota superior para la discrepancia es la mejor posible para esos números.

6.1. Máxima discrepancia aceptable por nuestro algoritmo

La Figura 6.1 muestra la discrepancia máxima aceptable por el Algoritmo 1 con todas las secuencias de entrada. Aquí puede apreciarse que la máxima discrepancia simple aceptable por nuestro algoritmo es muy similar para todas las entradas. Sospechamos que la forma de escalera que muestra el gráfico tiene que ver principalmente con una pérdida de precisión por el uso de floats.

El valor de la discrepancia máxima aceptable en el paso i puede acotarse usando el Lemma 2.5 en [6] que prueba lo siguiente para todo conjunto de bloques de dígitos $\{u_1, u_2, \ldots, u_i\}$:

$$D(u_1 u_2 \ldots u_i, b) \leqslant \sum_{j=1}^{i} \varepsilon_j |u_j| \Bigg/ \sum_{h=1}^{i} |u_h| \,,$$

poniendo como u_j el bloque de caracteres agregados en el paso j en base b, de manera que $|u_j|$ resulta ser la cantidad de dígitos agregados en el paso j.

Los resultados empíricos muestran que las secuencias generadas por nuestro algoritmo presentan una discrepancia muy inferior a este máximo.

La diferencia entre la discrepancia máxima aceptable y la discrepancia medida en los experimentos es de alrededor de un orden de magnitud. La relación entre máximo y valor observado es similar en todas las bases que llega a revisar el algoritmo. La Figura 6.2 muestra el caso de Champernowne, pero para todas las entradas el resultado es similar.

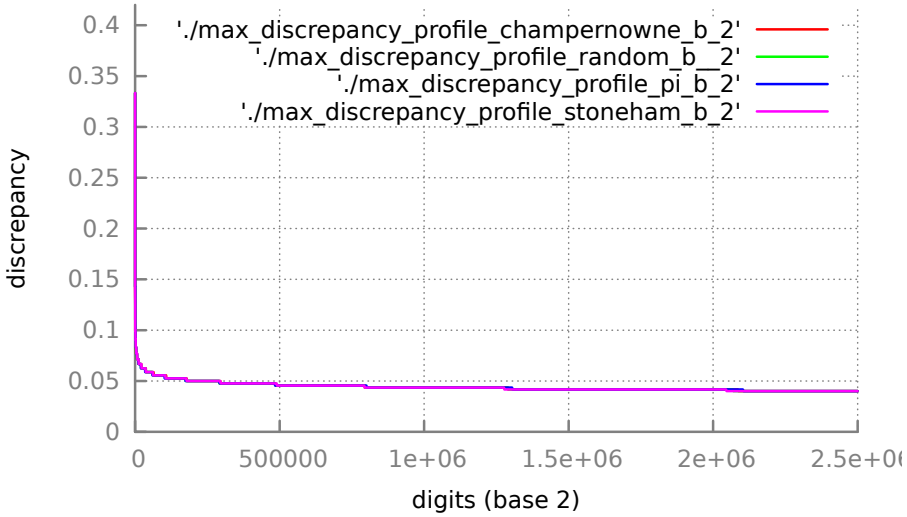

Figura 6.1: Discrepancia máxima aceptable por el algoritmo para todas las salidas. Eje X: discrepancia simple. Eje Y: dígitos del bloque de entrada, la linea muestra cual es la discrepancia máxima en los segmentos iniciales para cada longitud en base 2

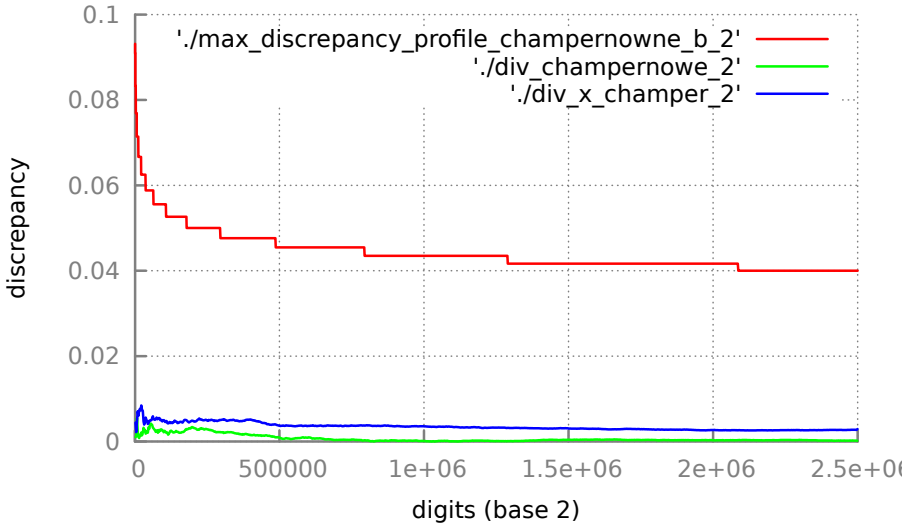

Figura 6.2: Discrepancia máxima de Champernowne, discrepancia observada de Champernowne, discrepancia observada de $x(Champernowne)$. Eje X: discrepancia simple. Eje Y: dígitos del bloque de entrada, la línea muestra cual es la discrepancia máxima y la observada en los segmentos iniciales para cada longitud

6.2. Discrepancia de las secuencias de entrada y las de salida

Graficamos la discrepancia simple de prefijos largos de la expansión de los números obtenidos por nuestros algoritmo, en diferentes bases, la función $D(u, b)$. Comparamos con los segmentos iniciales de las secuencias utilizadas como entrada.

Para visualizar la discrepancia de las secuencias de entrada y salida creamos gráficos de discrepancia simple (en diferentes bases) en función de la longitud de segmentos iniciales de las distintas secuencias en diferentes conjuntos de bases. Los conjuntos elegidos fueron las potencias de 2 menores a 25 (i.e. bases 2, 4, 8 y 16) y los primeros seis enteros multiplicativamente independientes (es decir, las bases 2, 3, 5, 6, 7 y 10). Estos gráficos son las Figuras 6.3 a 6.20.

En el eje X se ve la cantidad de dígitos de los segmentos iniciales medidos en base 2. Para las otras bases se buscó la posición equivalente tras realizar el cambio de base. En el eje Y se mide la discrepancia simple. Las líneas muestran la discrepancia simple en las diferentes bases de los segmentos iniciales de los bloques.

Tengamos en mente que si un número es normal en una base, también es normal en todas las potencias de esa base; es decir, en todas las bases multiplicativamente dependientes [17, 22]. Teniendo esto en mente, es esperable que las discrepancias simples para potencias de una base se comporten de forma similar entre sí, en caso de que la secuencia sea normal en esa base. Sin embargo, para bases que son multiplicativamente independientes el comportamiento no es predecible. Como mostraron Becher and Slaman [8] las funciones de discrepancia para las bases multiplicativamente independientes son independientes entre sí. Los gráficos de los segmentos iniciales podrían revelar esta falta de correlación.

En efecto, puede apreciarse en la Figura 6.11 como las discrepancias simples de 4 bases multiplicativamente dependientes entre sí tienen curvas muy similares. Tienen picos aproximadamente en los mismos puntos del eje X y decrecen con pendientes similares. Mientras tanto en gráficos como la Figura 6.10 la relación entre las diferentes curvas, que representan discrepancias en bases multiplicativamente independientes, no parece existir.

6.3. Comparación de las discrepacias computadas

Además para las bases 2, 6 y 10 generamos gráficos de las discrepancias de los segmentos iniciales para todas las secuencias estudiadas. Elegimos la base 2 porque la entrada es analizada en base 2, y las bases 6 y 10 porque los bloques de entrada del número de Stoneham $\alpha_{2,3}$ y el número de Champernowne respectivamente tienen comportamientos peculiares en esas bases. Podemos ver la forma particular de la discrepancia de estas secuencias en sus respectivos gráficos (ver Figura 6.4 para Champernowne y Figura 6.13 para Stoneham). Los gráficos que muestran estas secuencias se muestran dos veces a diferentes escalas para que se puedan ver claramente.

Los gráficos son iguales a los de discrepancia simple para potencias de 2 y para bases multiplicativamente independientes, con la única salvedad de la discrepancia de qué números muestran.

Para todos los gráficos la discrepancia fue medida en 1000 segmentos finales con posiciones finales equidistantes.

Como era esperado, la discrepancia en las secuencias de salida y en los números que sospechamos absolutamente normales (π, Champernowne, Stoneham $\alpha_{2,3}$ en bases distintas a 6, y en la secuencia pseudoaleatoria) tiende a decrecer. Creemos que en el límite la discrepancia de estos números es cero. Para Champernowne en base 10, para Stoneham $\alpha_{2,3}$ en base 2 y

en base 3, y para la salida de nuestro programa en todas las bases, esta convergencia esta demostrada.

Desde el punto de vista teórico, para nuestro algoritmo, la magnitud de la base no tiene una relación directa con la magnitud de la discrepancia. Sin embargo, observamos que la discrepancia en base 2 de las secuencias de salida generadas por nuestro algoritmo mayoriza casi siempre a las demás. En consecuencia vemos claramente en la Figura 6.21 que para el caso de base 2 la discrepancia de las secuencias generadas por nuestro algoritmo es bastante mayor que la de las secuencias de entrada. En las Figuras 6.22 y 6.24 notamos que ese efecto no existe o es mucho menos marcado para las bases 6 y 10. En contraste, las secuencias de entrada tienen discrepancia menor en base 2.

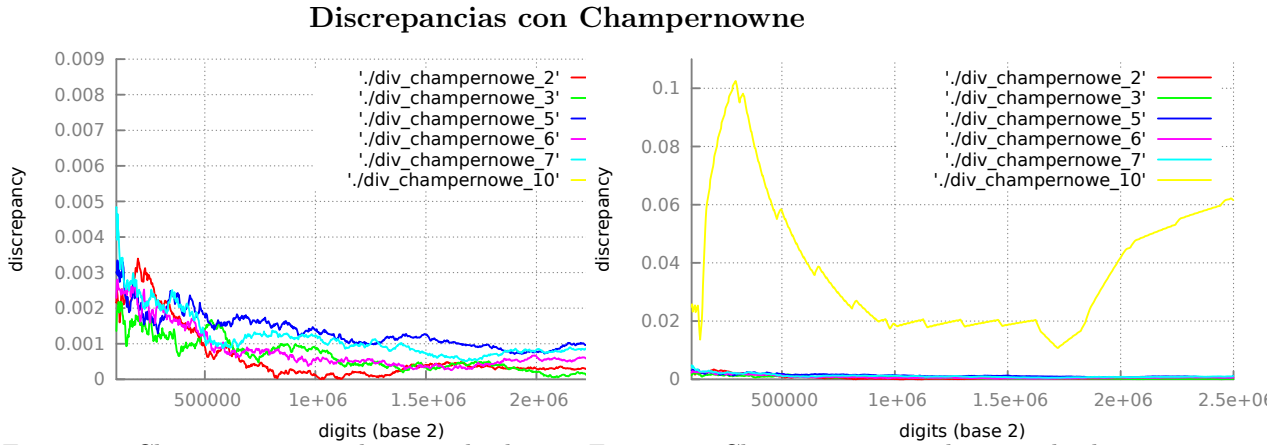

Figura 6.3: Champernowne en bases multiplicativamente independientes hasta base 7.

Figura 6.4: Champernowne en bases multiplicativamente independientes, hasta base 10

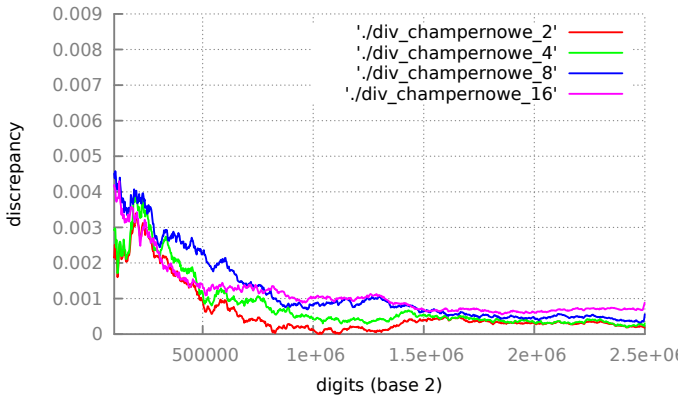

Figura 6.5: Champernowne en bases potencias de dos

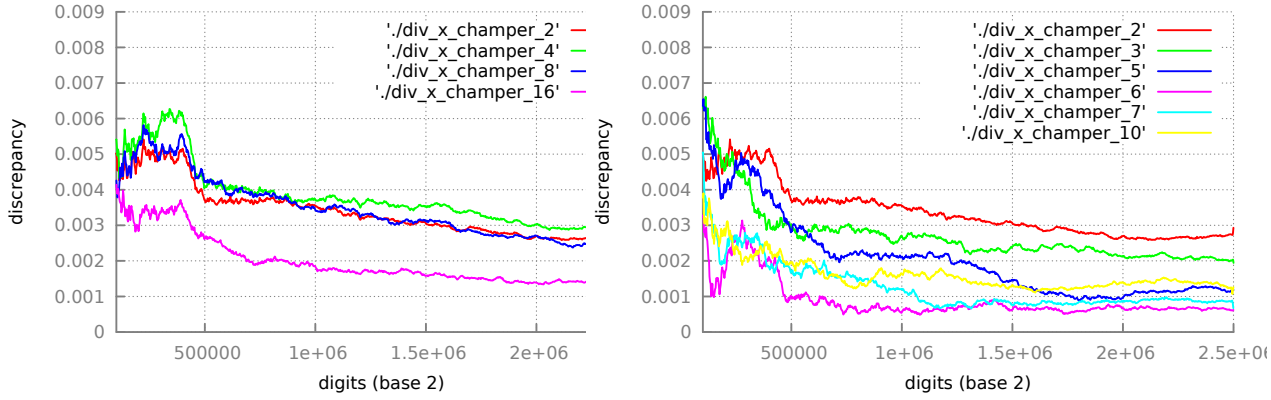

Figura 6.6: $x(Champernowne)$ bases potencias de dos

Figura 6.7: $x(Champernowne)$ en bases multiplicativamente independientes

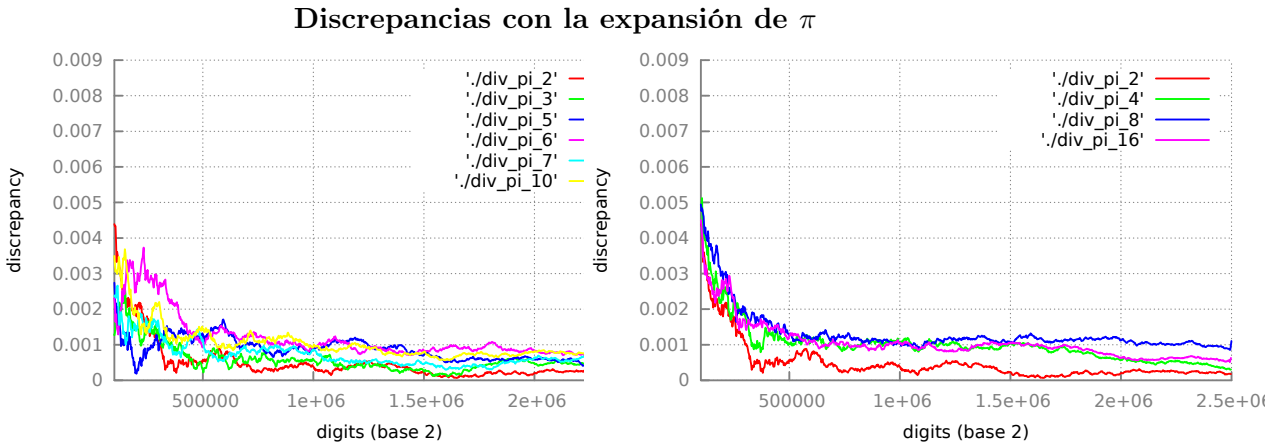

Figura 6.8: π en bases multiplicativamente independientes

Figura 6.9: Discrepancia de π en bases potencias de dos.

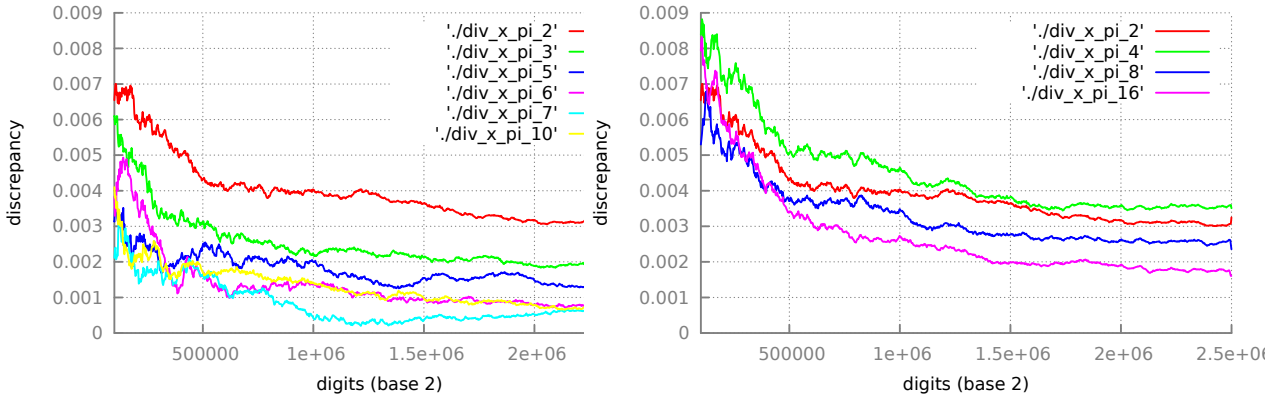

Figura 6.10: $x(\pi)$ en bases multiplicativamente independientes

Figura 6.11: Discrepancia de $x(\pi)$ en bases potencias de dos

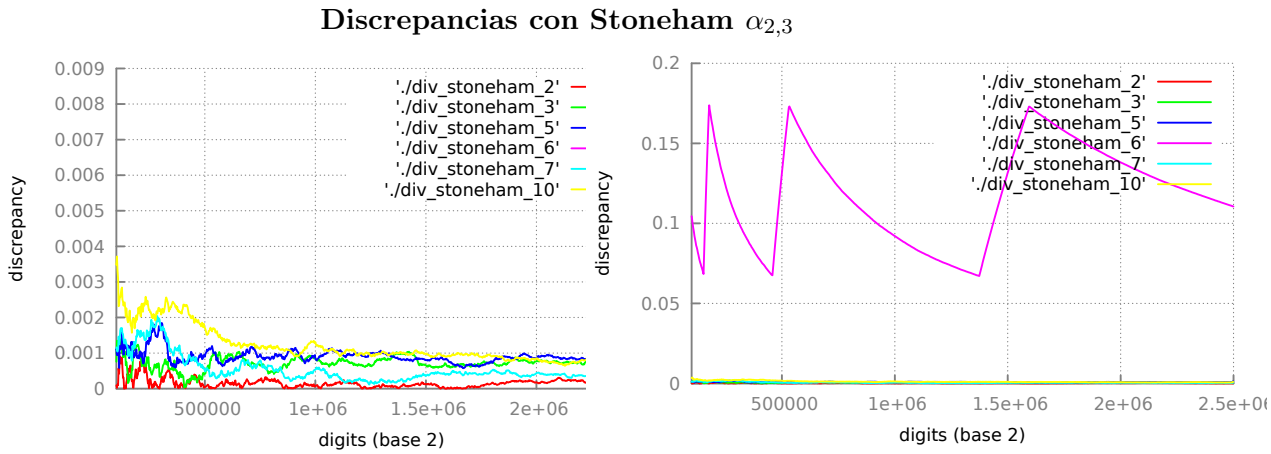

Figura 6.12: Stoneham $\alpha_{2,3}$ en bases multiplicativamente independientes

Figura 6.13: Stoneham $\alpha_{2,3}$ en bases multiplicativamente independientes

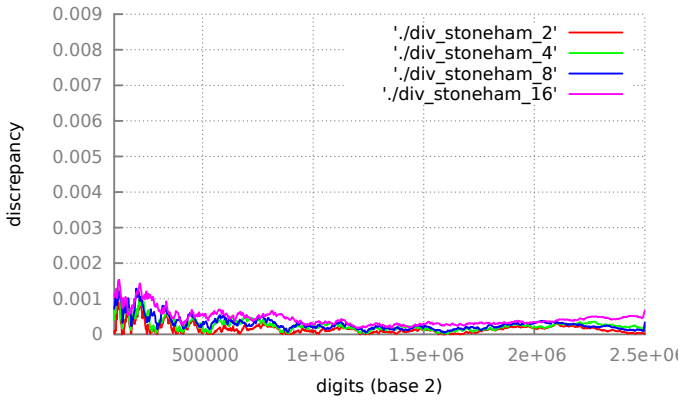

Figura 6.14: Stoneham $\alpha_{2,3}$ en bases potencias de dos

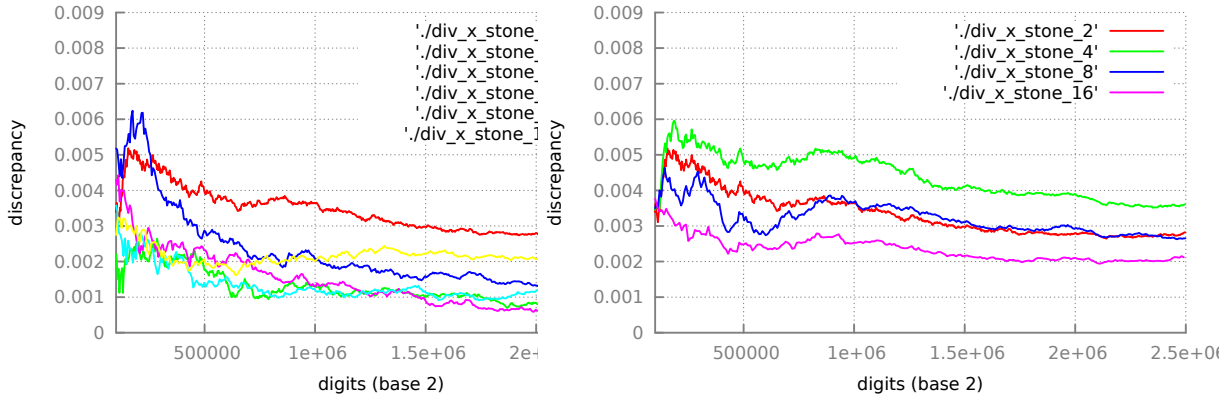

Figura 6.15: x(Stoneham $\alpha_{2,3}$) bases multiplicativamente independientes

Figura 6.16: x(Stoneham $\alpha_{2,3}$) en bases potencias de dos

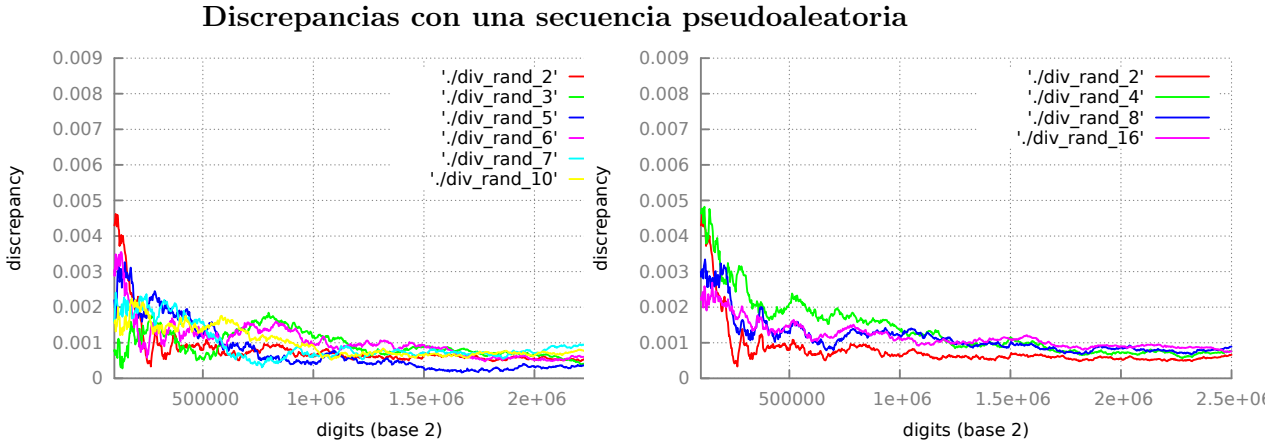

Figura 6.17: Pseudoaleatorio bases multiplicativamente independientes

Figura 6.18: Pseudoaleatorio en bases potencias de dos

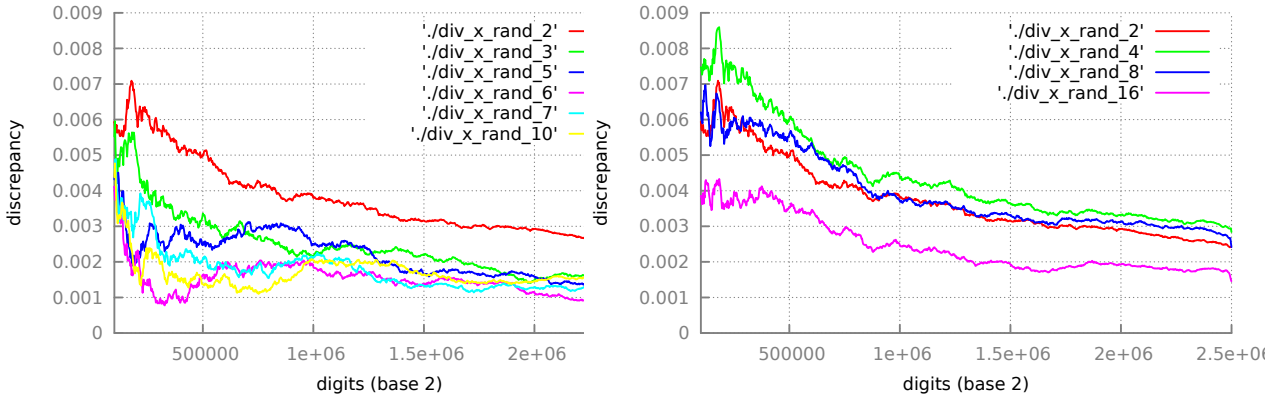

Figura 6.19: x(pseudoaleatorio) en bases multiplicativamente independientes

Figura 6.20: x(pseudoaleatorio) bases potencias de dos

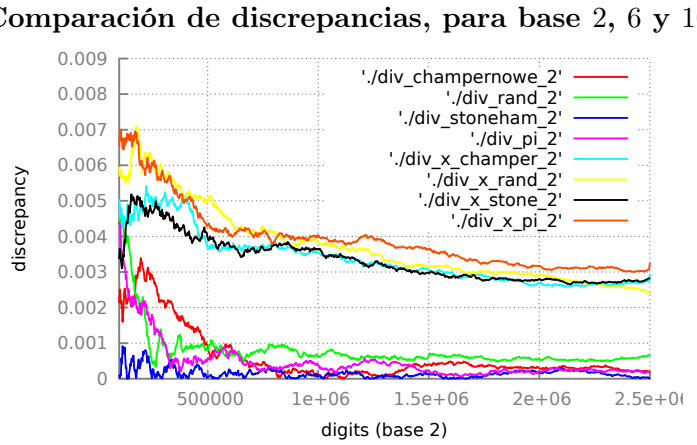

Figura 6.21: Base 2, cuatro inputs y cuatro outputs

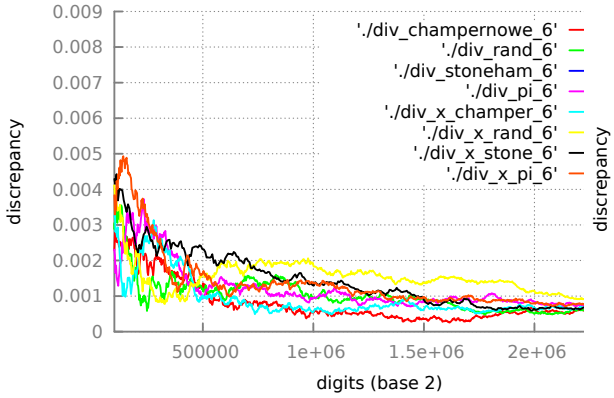

Figura 6.22: Base 6, cuatro inputs y tres outputs (falta Stoneham $\alpha_{2,3}$)

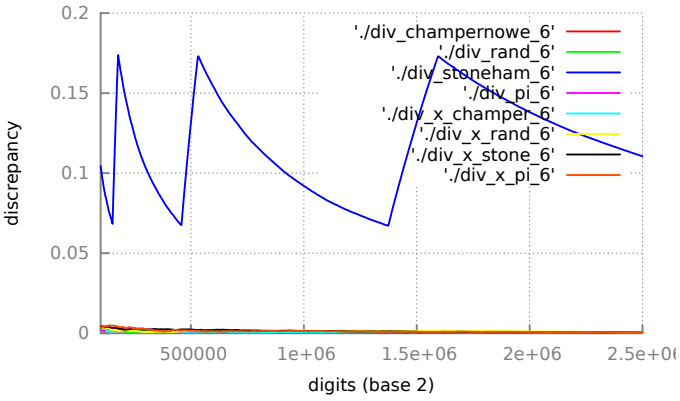

Figura 6.23: Base 6, cuatro inputs y cuatro outputs

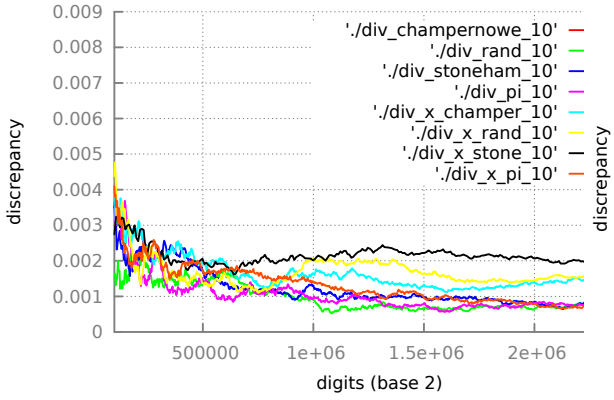

Figura 6.24: Base 10, cuatro inputs y tres outputs (falta Champernowne)

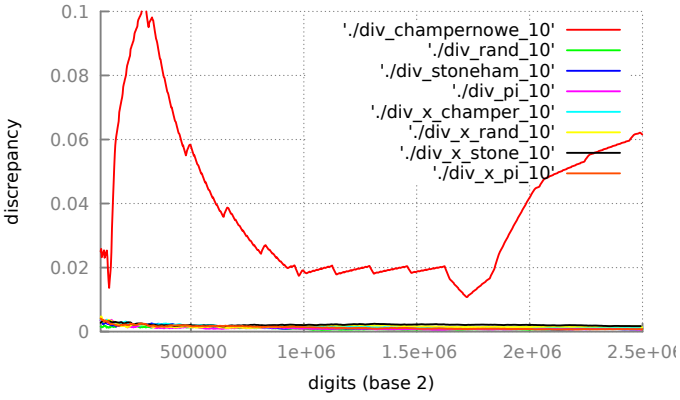

Figura 6.25: Base 10, cuatro inputs y cuatro outputs.

Bibliografía

[1] Francisco Aragon Artacho, David H. Bailey, Jonathan M. Borwein, and Peter B. Borwein. Walking on real numbers. *The Mathematical Intelligencer*, 35(1):42–60, March 2013.

[2] David H. Bailey and Jonathan M. Borwein. Nonnormality of Stoneham constants. *Ramanujan Journal.*, 29(1-3):409–422, 2012.

[3] David H. Bailey and Richard E. Crandall. On the random character of fundamental constant expansions. *Experimental Mathematics*, 10(2):175–190, 2001.

[4] Verónica Becher and Santiago Figueira. An example of a computable absolutely normal number. *Theoretical Computer Science*, 270(1-2):947–958, 2002.

[5] Verónica Becher, Santiago Figueira, and Rafael Picchi. Turing's unpublished algorithm for normal numbers. *Theoreical Computer Science*, 377(1-3):126–138, 2007.

[6] Verónica Becher, Pablo Ariel Heiber, and Theodore A. Slaman. Normal numbers and the borel hierarchy. *Fundamenta Matemathicae*, 2013. In press.

[7] Verónica Becher, Pablo Ariel Heiber, and Theodore A. Slaman. A polynomial-time algorithm for computing absolutely normal numbers. *Information and Computation*, 232:1 – 9, 2013.

[8] Verónica Becher and Theodore Slaman. On the normality of numbers to different bases. Submitted, 2013.

[9] É. Borel. Sur les chiffres décimaux de $\sqrt{2}$ et divers problèmes de probabilités en chaînes. *Comptes Rendus de l'Académie des Sciences de Paris*, (230):591—-593, 1950.

[10] Émile Borel. Les probabilités d'enombrables et leurs applications arithmétiques. *Supplemento di Rendiconti del circolo matematico di Palermo*, 27:247–271, 1909.

[11] Yann Bugeaud. *Distribution Modulo One and Diophantine Approximation*. Number 193 in Cambridge Tracts in Mathematics. Cambridge University Press, Cambridge, UK, 2012.

[12] D. G. Champernowne. The construction of decimals normal in the scale of ten. *Journal of the London Mathematical Society*, 8:254–260, 1933.

[13] H. Davenport and P. Erdös. Note on normal decimals. *Canadian J. Math*, pages 58–63, 1952.

[14] I.S. Gal and L. Gal. The discrepancy of the sequence $(2^n x)$. *Indagationes Mathematicae*, 26:129–143, 1964.

[15] L. Kuipers and H. Niederreiter. *Uniform distribution of sequences*. Dover Publications, Inc., New York, 2006.

[16] M. B. Levin. On absolutely normal numbers. *Vestnik Moskovskogo Universiteta. Seriya Matematiki, Mekhaniki*, 87:31–37, 1979. (in Russian). English translation in Moscow University Mathematics Bulletin 34 (1979), 32-39.

[17] Mordechay Levin. On the discrepancy estimate of normal numbers. *Acta Arithmetica*, 88(2):99–111, 1999.

[18] John E. Maxfield. Normal k-tuples. *Pacific Journal of Mathematics*, 3:189–196, 1953.

[19] Johan Sejr Brinch Nielsen and Jakob Grue Simonsen. An experimental investigation of the normality of irrational algebraic numbers. *Mathematis of Computation*, 82:1837–1858, 2013.

[20] Paul Pollack and Joseph Vandehey. Some normal numbers generated by arithmetic functions. ArXive 1309.7386, 2013.

[21] J. Schiffer. Discrepancy of normal numbers. *Acta Arithmeticae*, 47:175–186, 1986.

[22] Wolfgang Schmidt. Über die Normalität von Zahlen zu verschiedenen Basen. *Acta Arithmetica*, 7:299–309, 1961/1962.

[23] Wolfgang M. Schmidt. On normal numbers. *Pacific Journal of Mathematics*, 10:661–672, 1960.

[24] Alan Turing. A note on normal numbers. In editor J.L.Britton, editor, *Collected Works of A.M. Turing: Pure Mathematics*, pages 117–119. North Holland, Amsterdam, 1992. with notes of the editor in 263–265.