La Columna de Matemática Computacional

Sección a cargo de

Tomás Recio

El objetivo de esta columna es presentar de manera sucinta, en cada uno de los números de La Gaceta*, alguna cuestión matemática en la que los cálculos, en un sentido muy amplio, tengan un papel destacado. Para cumplir este objetivo el editor de la columna (sin otros méritos que su interés y sin otros recursos que su mejor voluntad) quisiera contar con la colaboración de los lectores, a los que anima a remitirle (a la dirección que se indica al pie de página*[1]*) los trabajos y sugerencias que consideren oportunos.*

En este número ...

... presentamos un curioso artículo de los profesores de la Universidad de La Rioja, Óscar Ciaurri y Juan Luis Varona, sobre los errores matemáticos que aparecen al tratar de resolver algunos problemas mediante ciertos programas de Cálculo Simbólico, como *Mathematica*. Es de justicia señalar que Ciaurri y Varona nos advierten desde el principio que la elección de este conocido programa se debe a su familiaridad con el mismo, pero que seguramente, "mutatis mutandis", podrían haber mostrado fallos similares en otros programas de este tipo. El artículo comienza con una amena y documentada reflexión general sobre los "fallos" del ordenador, para proseguir con la descripción (e intento de explicación) de una colección de errores en este programa de cálculo (pretendidamente) exacto. Errores de diverso calibre (grado de alejamiento de la verdad), calado (probabilidad de tropezar con ese error en la práctica –pero, ¿la práctica de quién?)–, origen (causa del error), procedencia (descubierto por los propios autores, comunicado por otros,...), etc.

Como conclusión, los autores siguen manifestando su confianza en estos programas de Cálculo Simbólico, pero nos instan a usarlos desde el conocimiento de las matemáticas y la algorítmica involucradas. Dicen los autores: "Recomendamos al lector que sea precavido, pero no alarmista".

Amén.

[1]Tomás Recio. Departamento de Matemáticas. Facultad de Ciencias. Universidad de Cantabria. 39071 Santander. recio@matesco.unican.es

¿Podemos fiarnos de los cálculos efectuados con ordenador?[2]

por

Óscar Ciaurri y Juan Luis Varona[3]

En este artículo analizamos la fiabilidad de los cálculos hechos con ordenador. Es bien conocido que, si hemos usado algoritmos numéricos y aritmética de coma flotante, los resultados pueden estar afectados por diversos problemas de estabilidad y pérdida de precisión. Pero, ¿qué ocurre con los cálculos simbólicos efectuados por medio de los potentes programas de álgebra computacional de los que ahora disponemos? Por medio de numerosos ejemplos, y particularizando en el programa *Mathematica*, vemos que abundan los fallos, así como los comportamientos distintos de lo razonablemente esperado. Estos problemas son difíciles de detectar, y sólo los conocimientos matemáticos del usuario pueden ayudar a paliar sus efectos.

A todos nos han puesto alguna vez excusas del tipo ≪no ha podido ser por culpa del ordenador≫. Así que, antes de comenzar, aclaremos que no vamos aquí a caer en ello. Los pobres ordenadores no tienen la culpa de nada; son las personas que han escrito los programas quienes consiguen que funcionen de una manera u otra. Algunas veces, a nuestra entera satisfacción; otras, no tanto.

Pero el caso es que ahí tenemos a nuestro ordenador y a sus programas, que son los que interactúan con nosotros. Generalmente, el programador es, para el usuario, totalmente desconocido. Como si no existiera. No podemos plantearle ninguna aclaración. Tenemos que confiar (o no) en lo que el ordenador nos dice.

Y la realidad es que los ordenadores se introducen cada vez más en nuestras vidas. Comprobar que el *software* funciona como estaba previsto que lo hiciera es una tarea ardua, y su verificación formal es prácticamente imposible. Los programas que no incorporan ningún *bug*[4] son muy raros. Fallos tontos

[2]Mathematics Subject Classification (2000): Primary 68-01; Secondary 65–01.

[3]La investigación de los autores está subvencionada, en parte, por el proyecto BFM2003-06335-C03-03 de la DGI.

[4]Se acostumbra a denominar *bugs* (en inglés, 'bichos') a los errores en los programas. La historia cuenta que, en los primeros tiempos de la informática, cierto fallo de funcionamiento en un ordenador se debía a que en sus circuitos se había introducido un insecto. Este hecho popularizó la palabra. En la entrada '*Computer bug*' de Wikipedia (`http://en.wikipedia.org/wiki/Computer_bug`) se pueden encontrar más detalles sobre la etimología del término (¡incluso una foto del insecto!), junto con información adicional y otras anécdotas.

pueden dar lugar a catástrofes. Muy conocido es el mal funcionamiento de ciertos misiles defensivos *Patriot* durante la Guerra del Golfo por no usar números de suficiente precisión; esto supuso, el 25 de febrero de 1991, la muerte de 28 soldados norteamericanos en Arabia Saudí. O la explosión, el 4 de junio de 1996, del primer cohete *Ariane* 5 que se lanzaba; el cohete pertenecía a la Agencia Espacial Europea, y en su desarrollo se habían empleado 10 años y $7 \cdot 10^9$ dólares. El problema fue similar: una confusión entre números enteros y reales en el programa informático que controlaba el lanzamiento[5]. En [11] podemos encontrar muchos otros ejemplos, así como un análisis de las medidas que se intentan tomar para evitar todo tipo de fallos informáticos (lamentablemente, muchas veces las medidas sólo fueron propuestas, pero no llevadas a la práctica). También son muy interesantes las reflexiones de [9]; pese al tiempo transcurrido y a la rápida evolución de la informática, no han perdido nada de actualidad.

No sólo el *software* nos puede dar quebraderos de cabeza. También pueden fallar los circuitos electrónicos, el denominado *hardware*. El ejemplo más conocido es el del error en los primeros microprocesadores *Pentium*, que dividían mal determinadas combinaciones de números. No todo el mundo sabe que fue un matemático –Thomas R. Nicely– quien descubrió el fallo. En 1994, con ayuda de varios ordenadores, Nicely estaba intentando estimar la suma de los inversos de los números primos gemelos[6], pero le aparecían resultados inconsistentes. Tras arduos esfuerzos, logró darse cuenta de que el microprocesador no dividía bien (véase [10]). Como consecuencia, el 20 de diciembre de 1994, Intel, la multinacional fabricante del *Pentium*, ofreció reemplazar, gratis, todos los chips defectuosos.

En lo que a nosotros –matemáticos– concierne, los ordenadores ya desempeñan un papel fundamental para ayudarnos a realizar cálculos, tanto numéricos como simbólicos. Tal es así que, entre otros colectivos, hay gente tentada a opinar que aprender matemáticas es cada vez menos importante, pues los ordenadores hacen ≪todo≫ lo necesario. La vieja polémica de las calculadoras se ha hecho mayor. De hecho, habitualmente, los ordenadores operan o representan gráficas mejor que cualquier matemático experto; y en las tareas repetitivas no se aburren. Pero no hay que alarmarse; nuestra profesión no va

[5]No siempre se puede achacar a la informática el fracaso de las misiones espaciales: el 23 de septiembre 1999, la sonda *Mars Climate Orbiter* se estrelló contra Marte. La explicación que dio la NASA es que una empresa involucrada en la construcción de la sonda había suministrado los datos requeridos en unidades británicas (millas, libras...), no en unidades del sistema internacional (kilómetros, kilogramos...) como la NASA esperaba. Contrariamente a nuestra opinión, hay quienes también echan la culpa a la informática de esto, y lo ponen como ejemplo de *bug* desastroso; pero, ¿qué podía hacer el programa que controlaba el descenso de la sonda –por muy bien hecho que estuviera– si los datos que tenía que usar estaban equivocados de antemano?

[6]No se sabe si hay o no infinitos de ellos, pero sí que la suma de sus inversos es finita (por el contrario, la serie de los inversos de los números primos es divergente).

a desaparecer. Es realmente difícil que alguien logre indicarle al ordenador que le resuelva algún problema si ni siquiera sabe plantear lo que quiere, ni darle las instrucciones oportunas. Por el contrario, si sabe hacerlo, es que *sabe matemáticas*; en ese caso, hará muy bien en servirse del ordenador para que le ayude.

Aún así, todavía debemos estar preocupados. Los programas de cálculo actuales son impresionantes. Sus posibilidades son grandiosas, y el tiempo que nos ahorran podemos emplearlo –con gran satisfacción– en realizar las múltiples e imprescindibles tareas burocráticas a las que nuestra profesión nos obliga, y en rellenar el currículum en diecisiete formatos diferentes. Aunque hay un pero: ¿hasta qué punto podemos fiarnos de las respuestas que nos da un ordenador? Lamentablemente, nuestra experiencia nos hace ser algo pesimistas.

¿O quizás tenemos que mostrarnos contentos? En realidad, lo que observamos es que debemos ser muy precavidos con las respuestas de un ordenador. Ante un problema numérico, hay que tener mucho cuidado con aceptar lo que un programa informático afirma sin pensar qué puede estar haciendo por dentro: quizás ha podido emplear un algoritmo que no es estable, por ejemplo. Pero no sólo las respuestas numéricas son propensas a errores; también los hemos observado en las simbólicas. En general, de estas últimas suele ser más difícil darse cuenta, por inesperadas. Hace falta saber bastantes matemáticas, conocer el funcionamiento interno de los ordenadores, el tipo de algoritmos que suelen usar para responder a lo que le hayamos planteado, dónde un programador puede estar pasando algo por alto, si la respuesta es o no razonable ... También son necesarias ciertas dosis de intuición. Todo esto sólo se consigue con conocimientos matemáticos y experiencia. Definitivamente, las máquinas no nos suplantan.

Pero –hablando más en serio– no deja de ser desalentador; por muy expertos y cuidadosos que seamos, a veces es muy difícil detectar errores imprevisibles. Los fallos de *hardware* parecen menos peligrosos; hay mucho menos *hardware* diferente que *software*, luego el control de calidad es mucho mayor (además, los errores serios son fácilmente detectables pues el ordenador falla estrepitosamente). Por si queda alguno, hay una solución relativamente sencilla: podemos ejecutar el mismo programa en un ordenador con una circuitería distinta[7].

Otro tipo de problemas que muchos considerarán de ciencia ficción es la interacción de los rayos cósmicos. Al colisionar con los chips de memoria del ordenador, pueden alterar un *bit*, que quizás tenga influencia en lo que el ordenador está haciendo[8]. Esto es muy improbable, pero casi seguro que un

[7]En particular, esto –y muchos otros argumentos similares– es una razón de peso en contra de los monopolios informáticos.

[8]Todavía ningún sistema operativo lo ha usado como excusa para justificar sus cuelgues, pero llegará el día ...

proyecto que involucre varios años de uso de CPU se habrá topado con ello. Normalmente, esto no afecta a nuestro quehacer diario pero, si fuese necesario, repitiendo cálculos se solventa el problema.

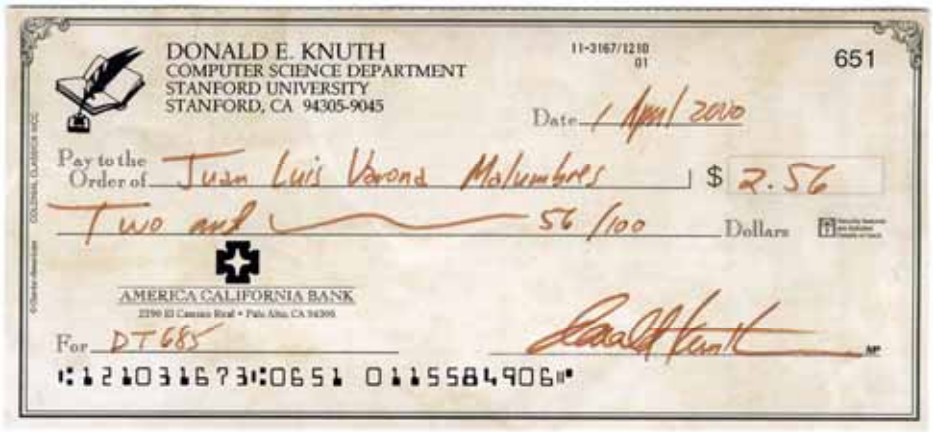
Los fallos más dañinos son los de *software*. Si un paquete destinado a una determinada tarea la hace mal por estar mal diseñado, la hará igual de mal en todo tipo de ordenadores (y esto, si es que existen versiones para más de una plataforma informática, claro). Chequear lo que estamos obteniendo por medio de otro paquete distinto no es fácil, y requiere mucho esfuerzo; en particular, habrá que aprender otra sintaxis para lograr plantear nuestro problema en el ordenador.

Todos estamos –o deberíamos estarlo– prevenidos contra los posibles errores que cualquier método numérico acarrea. En primer lugar, la aritmética real finita que usan tales métodos hace que muchos números debamos manejarlos de manera aproximada; hasta tal punto es así que esto es causa de que las operaciones básicas de sumar y multiplicar no conserven sus propiedades habituales (conmutatividad, asociatividad y distributividad). Además, los procesos infinitos del análisis matemático hay que abreviarlos, lo cual contribuye a introducir nuevos errores. También aparecen problemas de inestabilidad, que no siempre son fáciles de detectar ni analizar. No seguiremos por esta senda; simplemente, somos conscientes de las dificultades y esto nos hace estar alerta. No ocurre así cuando estamos tratando con los actuales programas que incluyen cálculo simbólico –también llamados sistemas de álgebra computacional–. Al fin y al cabo, no usan una aritmética finita, y tampoco deberían emplear algoritmos aproximados, sino exactos. Cualquiera podría pensar que, ahora, los errores no deberían existir. Si los hay, es fácil que pillen al usuario con la guardia baja.

Por otra parte, existen programas numéricos libres[9] de calidad contrastada, y también magníficos paquetes simbólicos especializados en diversos campos de la matemática. Pero no es así con los programas simbólicos de propósito general, que –salvo honrosas excepciones que no llegan al nivel de desarrollo de los líderes– están comercializados por diversas empresas. Esto quiere decir dos cosas: En primer lugar, lo habitual es que sean carísimos[10]. En segundo lugar, no podemos ver su código fuente, y por tanto no podemos saber cómo funcionan. Para nosotros, son ≪cajas negras≫, les metemos unos datos y nos

[9]En inglés, la palabra *free* que describe a estos programas significa tanto 'libre' (en el sentido de que podemos coger el código fuente del programa y hacer con él lo que queramos, dentro de ciertos límites razonables que nos impone la licencia), como 'gratis' (¡no nos cuestan dinero!). Aunque a gran parte de los usuarios no les preocupa, la accesibilidad al código es fundamental dentro del concepto de ≪software libre≫; quizás incluso prevaleciendo sobre el precio.

[10]Esto no sólo es un problema para el investigador que quiere usarlos, que quizás tenga dinero para ello, sino sobre todo para el profesor que quiere emplearlos en clase y pretende que los alumnos los puedan instalar en sus ordenadores personales para utilizarlos como herramienta habitual. Nunca hemos encontrado una solución apropiada para solventar esta dificultad.


DONALD E. KNUTH
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305-9045

11-3167/1210
01

651

Date 1 April 2010

Pay to the Order of Juan Luis Varona Malumbres $ 2.56

Two and 56/100 Dollars

AMERICA CALIFORNIA BANK

For DT 685

⑆121031673⑆0651 0115584906⑈

Figura 1: Un cheque firmado por Donald Knuth.

proporcionan otros; si algo no va bien, es muy difícil averiguar el motivo, por muy expertos que seamos. Más aún, si descubrimos un fallo, no podemos hacer nada por arreglarlo, ni podemos conseguir que el fabricante repare su producto defectuoso y nos suministre uno nuevo que funcione[11]. Si somos afortunados, podemos lograr que el fabricante atienda a nuestras comunicaciones sobre el fallo; e incluso que intente arreglarlo en una versión posterior del producto. Eso sí, la nueva versión nos la cobrará de nuevo, y es fácil que el arreglo de una cosa haya estropeado otras trece (algo exclusivamente achacable a la mala suerte, naturalmente).

Posiblemente, los paquetes de cálculo simbólico genéricos son los únicos programas no libres que los matemáticos nos vemos forzados a utilizar[12], pues no tienen contrapartida libre. Al menos, no la tienen con una potencia y facilidad de uso similares. No sería mala idea que todas las mañanas implorásemos a Dios para que nos proporcionara el tan deseado sistema de álgebra computacional libre de propósito general; bueno, quizás alguien tenga un día una idea mejor[13].

[11] Es sorprendente que las leyes permitan esto a la industria del *software*. Tal como ya hemos comentado, no obró así Intel cuando se descubrió su fallo en el Pentium; ni lo hacen los fabricantes de coches si se detecta que un modelo que está en el mercado tiene un defecto, por citar otro ejemplo.

[12] Bueno, salvo cuando las diversas administraciones públicas nos obligan a usar ... , y mejor no seguir para evitar cabrearnos (¡uy!, perdón por la palabrota).

[13] Por supuesto, el paradigma de *software* libre que los matemáticos usamos a todas horas es TEX. Hace ya bastantes años que Donald Knuth –su autor– decidió dejar de actualizarlo, salvo para corregir posibles errores en el programa. Sus motivos eran que esto iba a permitir librarse de ellos y, además, aseguraría la compatibilidad futura de los documentos escritos con TEX. Así mismo, decidió ofrecer recompensas monetarias –cuyas cuantías crecían con

En este artículo vamos a mostrar varios fallos de funcionamiento, o comportamiento distinto al esperado, de uno de los paquetes de cálculo simbólico más extendidos: *Mathematica* [15]. Que nadie entienda que esto es una crítica a este programa. En realidad, es el manipulador algebraico que más usamos los autores, y por tanto el que conocemos con más profundidad, y el que más nos gusta; incluso hemos escrito algunos artículos en los que mostramos su utilidad para diversas tareas (véanse [12, 13]). El tipo de problemas que aquí presentamos se da en todos los paquetes informáticos similares (en [14] puede encontrarse una excelente comparativa de las habilidades matemáticas de siete de estos programas). Pero, como *Mathematica* es el que mejor conocemos, ha tenido la ≪mala suerte≫ de que nos cebemos con él; sirva esto de estímulo para que *Wolfram Research* –la empresa que lo desarrolla– lo continúe mejorando. Insistimos en que nuestra pretensión es concienciar al usuario de que, en cualquier caso, debe permanecer alerta, y proporcionarle ideas que le pueden resultar útiles para detectar los posibles cálculos erróneos efectuados por este tipo de *software*.

Eso sí, no vamos a llegar al nivel de alarma de Householder, uno de los pioneros del álgebra lineal numérica y destacada figura de la matemática aplicada, quien declaró en cierta ocasión que nunca volaría en un aeroplano que hubiera sido diseñado con la aritmética de punto flotante, insistiendo en que no se podía confiar en la exactitud de tales cálculos.

Varios de los fallos que aquí presentamos los han descubierto quienes esto escriben. Algunos nos han aparecido en situaciones de nuestro trabajo como matemáticos, y los hemos aislado hasta encontrar un ejemplo sencillo en el que se manifestara el funcionamiento incorrecto. En numerosas ocasiones los hemos comentado con diversos colegas; su reacción ha sido siempre una combinación de sorpresa y desencanto. Del mismo modo, otros ejemplos nos han sido proporcionados por otros matemáticos, o los hemos leído en algún foro de *internet*. Tanto a los que nos los han comunicado personalmente como a los que los han divulgado en la red se lo agradecemos, pero seguro que errábamos si intentábamos dar una relación de todos ellos (lo cual sería, por supuesto, un ≪metafallo≫).

Hemos analizado diversas versiones no muy antiguas de *Mathematica*; en concreto, las versiones 3.0, 4.1, 4.2, 5.0 y 5.1 (para ser más precisos, 3.0.1, 4.1.5, 4.2.1, 5.0.0 y 5.1.0; pero no hay ningún tipo de información de qué es lo que cambia cuando varía el tercer dígito, ni existe ninguna publicidad de tales lanzamientos, ni conocemos forma alguna de actualizar el programa cuando esto ocurre; así que no hemos tenido en cuenta este tercer dígito en el número de versión). No siempre todas las versiones se comportan igual cuando

potencias de 2– a los que encontraran algún *bug* en TEX. Actualmente, si alguien descubriera un nuevo error, recibiría 327.68 dólares. Donald Knuth también gratifica (con 256 céntimos de dólar o, como él lo llama, un ≪dolar hexadecimal≫) a quien halla erratas en sus libros. Numerosos ≪gazapólogos≫ y coleccionistas de cheques incobrados hemos detectado bastantes de ellos (véase la figura 1), que serán corregidos en posteriores ediciones de los libros.

se enfrentan al mismo problema. Es más, veremos que, en ocasiones, las más nuevas lo hacen peor.

A menudo usaremos la notación de *Mathematica* sin explicarla explícitamente. No hay ninguna necesidad de que el lector la conozca; ni siquiera de que haya manejado *Mathematica* alguna vez. Con un poco de buena voluntad, todo lo que aquí aparece será perfectamente comprensible por cualquiera que tenga conocimientos matemáticos de nivel preuniversitario o –para algunos ejemplos– de un primer curso universitario de cálculo.

1 Cálculo de límites

Desde la versión 5.0, *Mathematica* ya conoce la equivalencia de Stirling. Así, ya sabe que el límite

```
Limit[n^(n+1/2)*E^(-n)/n!, n -> Infinity]
```

vale $1/\sqrt{2\pi}$. Una buena mejora. Pero no queremos con esto decir que las versiones anteriores fueran malas porque no supieran calcularlo. No es ése el problema; si no saben y nos devuelven el límite sin hacer (como así ocurría), no nos engañan. Lo grave no es eso, sino que nos proporcione resultados erróneos.

Y es que eso también sucede. Hasta el punto que nos atrevemos a decir que el tratamiento que hace *Mathematica* de los límites es una chapuza. Sin paliativos. Si pensamos en funciones de variable real, todos sabemos que, para que exista el límite, tiene que haber límite por la derecha y por la izquierda. En cambio, *Mathematica* parece ignorarlo; así, por ejemplo, afirma alegremente que

```
Limit[Sin[x]/Abs[x], x -> 0]
```

vale 1. Y no acertamos a comprender por qué lo hace así, pues es capaz de calcular los dos límites laterales, que son 1 por la derecha y -1 por la izquierda. Esto se consigue, respectivamente, con la sintaxis

```
Limit[Sin[x]/Abs[x], x -> 0, Direction -> -1]
Limit[Sin[x]/Abs[x], x -> 0, Direction -> 1]
```

Aun así, podemos ser benevolentes. Si estamos advertidos de este fallo, no es un problema muy grave; basta que nos acordemos de que debemos calcular ambos límites laterales. ¿Pero qué pasa si no lo sabíamos?

2 Gráficos ficticios

Representando funciones, *Mathematica* es bastante bueno[14]. Por ejemplo, es capaz de dibujar la gráfica de $\text{sen}(x)/x$ sin dificultad. Sin embargo, cuando

[14] Aunque podría mejorar; otros paquetes son mucho más útiles para dibujar funciones implícitas en dos e incluso tres dimensiones.

la función tiene una discontinuidad de salto –como $\mathrm{sen}(x)/|x|$ o $\mathrm{tg}(x)$–, suele incluir en el salto una línea vertical inexistente, incluso aunque los saltos no sean finitos. Se lo podemos perdonar. Lo normal es que cualquier programa que dibuja funciones lo haga «dando valores» (como muchos alumnos); así, es difícil que detecte las discontinuidades. Quizás sería exigir demasiado pensar que un programa puede desenvolverse ante un concepto matemático como puede ser el de discontinuidad; podemos pues asumir que se lo teníamos que haber indicado de alguna manera.

Además, dar valores lo hace con una cierta «inteligencia». Por defecto, utiliza cierta cantidad de puntos equidistantes (su número depende de la versión de *Mathematica* que estemos usando); pero incluye un algoritmo de muestreo adaptativo que muchas veces le hace tomar puntos adicionales. Esto suele funcionar bastante bien, y ha ido mejorando con las versiones. Por ejemplo, el libro [5, § 4.2.2] nos muestra que *Mathematica* 1.2 y 2.0 dibujaban

```
Plot[x + Sin[2*Pi*x], {x, 0, 24}]
```

exactamente igual que la recta $y = x$. Internamente, para hacer ese gráfico, *Mathematica* tomaba los 25 puntos $(x_j, x_j + \mathrm{sen}(2\pi x_j))$, $x_j = 0, 1, 2, \ldots, 24$, y los unía; y da la casualidad de que todos ellos están sobre la recta $y = x$. Se puede solucionar el problema dando un número distinto de puntos de muestreo, aunque para eso tenemos que sospechar que algo falla. Funciona tanto asignar un número mayor (`PlotPoints -> 50`) como menor (`PlotPoints -> 20`). Pero los métodos de muestreo adaptativo de las versiones posteriores han mejorado mucho, y no hemos observado dibujos erróneos de ninguna función $y = f(x)$ razonable.

Lamentablemente, estos algoritmos adaptativos son bastante más difíciles de usar para funciones $z = f(x, y)$; y, sobre todo, consumirían mucho tiempo de cálculo. Así que los desarrolladores de *Mathematica* han optado por no aplicarlos. Como consecuencia, si la función que representamos tiene oscilaciones, existe la posibilidad de que obtengamos un dibujo que no se corresponde con la realidad. Así, por ejemplo, *Mathematica* 5.0 y 5.1 representan

```
Plot3D[y^2 + Sin[23*x], {x, 0, 2Pi}, {y, -1, 1}]
```

tal como aparece en la figura 2. La suavidad del dibujo no nos hace pensar que sea erróneo; pero si reflexionamos nos damos cuenta de que la parte `Sin[23*x]` debe forzosamente producir una oscilación que se ha perdido (y que se puede ver añadiendo un `PlotPoints -> 50`). Ha dado la casualidad de que los puntos de muestreo que ha usado *Mathematica* han estado todos ellos sobre una superficie bastante suave, y ésa es la que ha dibujado. En versiones anteriores (la 4.2, por ejemplo), *Mathematica* usaba un número distinto de puntos de muestreo, y el problema se observaba poniendo `13` o `15` en el lugar del `23`.

Descubramos el truco: este fallo lo hemos encontrado adrede. Nuestro conocimiento de lo que son capaces de hacer este tipo de programas nos

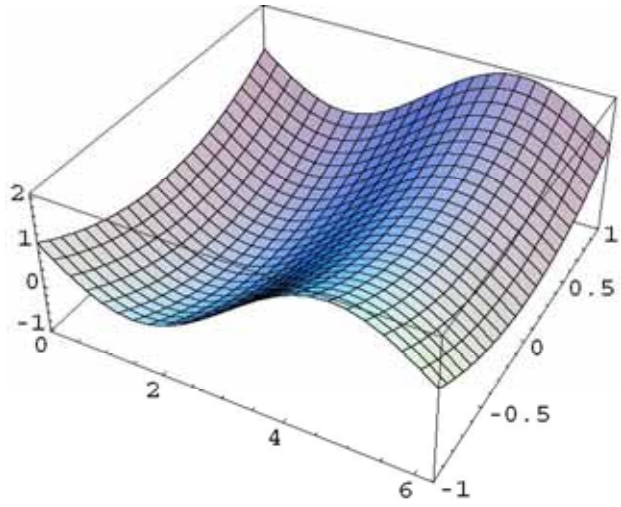


Figura 2: La supuesta gráfica de la función $z = y^2 + \text{sen}(23x)$ con $x \in [0, 2\pi]$, $y \in [-1, 1]$, según *Mathematica* 5.

hacía pensar que se tenía que producir. Pero sólo sucede cuando los puntos de muestreo están en determinados lugares estratégicos. La mejor manera de ayudar a la casualidad a que esto ocurra es usar un bucle:

```
For[k=1, k<=30, k++,
  Plot3D[Sin[k*x], {x, 0, 2Pi}, {y, -1, 1}]
]
```

Así es fácil encontrar un dibujo ficticio (en el ejemplo de la figura 2, el sumando y^2 no tiene importancia, pero hace que la función tenga realmente dos variables).

Como contrapartida, aprendemos una lección: cuando hagamos una gráfica, ante la menor sospecha conviene variar el número de `PlotPoints` asignados por defecto, y probar con varios valores. De este modo, podremos detectar una posible cancelación casual de la oscilación. No olvidemos que, aunque hemos confesado que esta vez el fallo lo hemos buscado aposta, puede producirse en un caso real que no nos resulte tan evidente.

3 Simplificar tiene su miga

Cuando le pedimos que simplifique una expresión, *Mathematica* es muy cuidadoso. Veámoslo por medio de un par de ejemplos. Si no lo pensamos demasiado, es habitual que nosotros mismos simplifiquemos $\operatorname{arcsen}(\operatorname{sen}(x)) = x$; pero esto no es cierto en general, sólo cuando $x \in [-\pi/2, \pi/2]$. *Mathematica* no pasa esto por alto, de tal manera que `Simplify[ArcSin[Sin[x]]]` devuelve $\operatorname{arcsen}(\operatorname{sen}(x))$, no x. Para conseguir que realmente simplifique, debemos especificar

```
Simplify[ArcSin[Sin[x]], -Pi/2 <= x <= Pi/2]
```

De manera similar, la orden `Simplify` tampoco hace nada si se la aplicamos a `Sin[x + 2*n*Pi]` (muchos matemáticos estamos tan acostumbrados a que n denote números enteros que damos por hecho que siempre es así, y posiblemente hubiéramos afirmado que $\operatorname{sen}(x + 2n\pi) = \operatorname{sen}(x)$ sin pestañear; pero *Mathematica* es prudente, y no cae en esa trampa). Para que simplifique, hay que indicarle que que n es un entero, lo que se consigue, por ejemplo, así:

```
Simplify[Sin[x + 2*n*Pi], Element[n, Integers]]
```

Además, es muy importante tener en cuenta que no siempre está claro qué queremos decir con «simplificar». ¿Qué es más simple, $(x^{100} - 1)/(x - 1)$ o los 100 sumandos que aparecen al dividir? ¿Y $\cos^2(x)$ o $(1 + \cos(2x))/2$? Pues depende de para qué lo queramos, y eso no lo puede saber la máquina. En consecuencia, todos los programas de cálculo simbólico tienen bastantes comandos destinados a efectuar muy diversas manipulaciones. No importa aquí cuáles son esos comandos, pero sí que conviene caer en la cuenta de que, si queremos que los cálculos que estemos efectuando avancen en la dirección que nos interesa, no nos queda otro remedio que conocerlos; no basta con servirse de una sola orden como `Simplify`.

Por otra parte, simplificar puede suponer muchísimo tiempo de cálculo. Para simplificar una expresión, *Mathematica* intenta aplicar lo que denomina reglas de transformación. Tiene que ver cuáles, de las muchas que tiene almacenadas, se ajustan a la expresión en cuestión, y adónde llevan. Es habitual que, ante una expresión medianamente complicada, haya multitud de caminos que se pueden explorar, y muy pocos conducen a buen puerto. Así, *Mathematica* dispone de dos órdenes, `Simplify` y `FullSimplify`. La segunda permite que el tiempo empleado sea mayor, y usa más reglas de transformación. Aunque, en nuestra opinión, hay veces que es incomprensible que el comando básico no se dé cuenta de algunas cosas que a cualquier matemático le parecen obvias. Por ejemplo, `Simplify` no se percata de que $\log(8)/\log(2)$ vale 3; si queremos conseguir esa simplificación, hay que hacerlo con `FullSimplify[Log[8]/Log[2]]`.

A veces, ni siquiera `FullSimplify` hace un trabajo que pudiéramos considerar satisfactorio. Es evidente que $\sum_{k=1}^{n} \sqrt{k} - \sum_{k=1}^{n-1} \sqrt{k} = \sqrt{n}$; sin embargo,

```
FullSimplify[
  Sum[Sqrt[k], {k, 1, n}] - Sum[Sqrt[k], {k, 1, n-1}]
]
```

no nos proporciona la esperada respuesta $\sqrt{n}$. Ni siquiera si le indicamos que n es un entero positivo, no vayamos a pensar que ése es el problema. Hay quien diría, en defensa de *Mathematica*, que el usuario puede añadir sus propias reglas de transformación, y de ese modo podríamos conseguir que simplificara, por ejemplo, la expresión anterior. No es tarea fácil y, además, ¡para ese viaje no hacían falta esas alforjas!

Otro ilustrativo ejemplo con el que las rutinas de simplificación no saben enfrentarse es

```
FullSimplify[12345678901!/12345678900!]
```

La respuesta debería haber sido 12345678901, pero *Mathematica* devuelve ese cociente sin simplificar. Y eso que sí que evalúa `FullSimplify[(n+1)!/n!]` como $n+1$. Vale la pena analizar lo que está pasando. *Mathematica* puede calcular factoriales, y eso intenta; si sabe calcularlo tal como está, ¿para qué simplificar?, debe pensar. Pero se da cuenta de que esos factoriales son realmente demasiado grandes (imagínese el lector cuantos dígitos podría tener 12345678901!), así que no los hace. Y ahí se acabó la historia; se percata de que la primera estrategia no vale, pero no usa la alternativa: simplificar antes de operar. ¿Cuántas veces le habremos dicho a nuestros alumnos que es mejor simplificar antes de ponerse a «echar cuentas» como locos?

4 Un par de simplificaciones erróneas

Pongámonos de nuevo del lado de *Mathematica*, y volvamos a mostrar ejemplos de la cautela que emplea al simplificar. Así, `Simplify[Sqrt[x^2]]` no devuelve x, pues se da cuenta de que x puede ser un número real negativo. Tampoco con `Simplify[Sqrt[x^4]]` obtenemos x^2 como respuesta, que no sería cierta si x es un complejo. Si no es éste el caso y queremos conseguir que simplifique, debemos indicárselo expresamente. A decir verdad, los investigadores que trabajan en temas relacionados con el cálculo simbólico son muy conscientes de las dificultades inherentes a la aritmética de números complejos (véase [6]), y procuran tenerlas en cuenta.

Esto no impide que, de vez en cuando, pueda surgir el habitual desliz. Aunque aquí no nos importa cómo están definidas, digamos que hay ciertas funciones que se conocen con el nombre de integrales elípticas (si al lector le ha picado la curiosidad de qué es eso, puede ojear [1, capítulo 17]; también hay algo de información en la propia ayuda de *Mathematica*, o en su manual [15]). Nosotros vamos ahora a usar dos de ellas, $F(\phi, m)$ y $K(m)$, que *Mathematica* representa con `EllipticF` y `EllipticK`. Pese a que la nota explicativa «qué es nuevo en la versión 5.1» no dice nada sobre ello, parece que esa versión

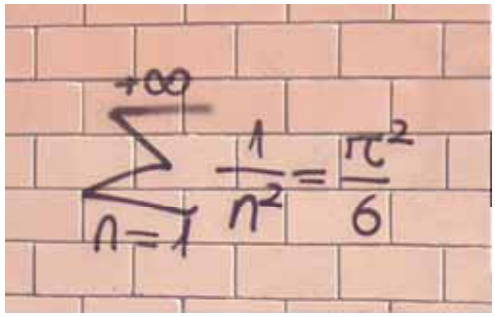


Figura 3: Fotografía tomada en el trayecto de un tren de cercanías valenciano.

ha añadido algunas rutinas para el tratamiento de estas funciones. Así, por ejemplo, *Mathematica* 5.1 afirma que

```
FullSimplify[EllipticF[I*Log[1 + Sqrt[2]], -1]]
```

(donde `I` es la unidad imaginaria) vale $-iK(-1)$, mientras que las versiones anteriores no transforman la función F en la K. Además, *Mathematica* (todas las versiones) sabe que `FunctionExpand[EllipticK[-1]]` vale $\Gamma(1/4)^2/(4\sqrt{2\pi}) \approx 1.31103$ (`FunctionExpand` es uno de los diversos comandos dedicados a simplificar a los que antes aludíamos). Como consecuencia de todo esto, *Mathematica* 5.1, ante la evaluación numérica

```
FullSimplify[EllipticF[I*Log[1 + Sqrt[2]], -1]] // N
```

nos proporciona el resultado $-1.31103i$. Por el contrario, si no hubiéramos simplificado, lo que obtenemos es que

```
EllipticF[I*Log[1 + Sqrt[2]], -1] // N
```

vale $1.31103i$. ¿Qué es lo que está pasando? Pues que $F(i*\log(1+\sqrt{2}), -1)$ no es $-iK(-1)$, sino $iK(-1)$. Aparentemente, al programador de *Mathematica* que estaba incluyendo las mejoras se le ha deslizado un signo «$-$» en alguna línea de código. Parece fácil de arreglar en futuras versiones. Veremos ...

Finalicemos esta sección analizando un error muy «original». Comencemos recordando que la serie $\zeta(s) = \sum_{n=1}^{\infty} 1/n^s$ es convergente cuando $s > 1$. Hay expresiones exactas de $\zeta(2k)$, con $k = 1, 2, 3, \ldots$, y *Mathematica* nos las proporciona con órdenes del tipo `Zeta[2k]`; obsérvese que, tal como se deduce de la figura 3, el valor de $\zeta(2)$ es realmente muy conocido.[15] Pero no hay nada parecido para $\zeta(2k+1)$; en consecuencia, *Mathematica* deja `Zeta[3]` –o

[15] Existen numerosas demostraciones diferentes de que $\sum_{n=1}^{\infty} 1/n^2 = \pi^2/6$ (véase [7], donde se muestran catorce), aunque ninguna de ellas cabía en la pared fotografiada.

similares– sin evaluar.[16] Menos habitual es la función $\mathrm{Li}_k(z) = \sum_{n=1}^{\infty} z^n/n^k$, que *Mathematica* denota como `PolyLog[k,z]`.

Centrándonos en el caso $k = 2$, y si tomamos un complejo z tal que $|z| = 1$, la serie $\sum_{n=1}^{\infty} z^n/n^2$ es absolutamente convergente, ya que está mayorada por $\sum_{n=1}^{\infty} 1/n^2$. Elijamos un complejo de módulo 1, como $z = e^i$. *Mathematica* no conoce ninguna expresión sencilla para `PolyLog[2,E^I]` (no la hay, hasta donde nosotros sabemos), así que lo devuelve sin evaluar; y lo mismo sucede si se lo pedimos junto con `Simplify`, `FullSimplify` o `FunctionExpand`. Eso sí, si le indicamos que nos dé su valor numérico aproximado lo hace, y nos dice que es $0.324138 + 1.01396i$. Hasta aquí, todo correcto. Lo sorprendente llega cuando, de manera simbólica, utilizamos `PolyLog` y `FunctionExpand` junto con la función `Zeta` (en un argumento s en el que *Mathematica* no sepa evaluar $\zeta(s)$). Da lo mismo que sea en una suma, en dos componentes de un vector, etc. Así, por ejemplo, supongamos que ejecutamos la orden

```
FunctionExpand[{Zeta[3], PolyLog[2, E^I]}]
```

Como cabía esperar, la primera parte queda sin evaluar; pero en la segunda obtenemos una respuesta totalmente falsa: `ComplexInfinity` (ya hemos comentado que la serie involucrada es convergente). ¿Por qué esta conjunción estropea las cosas, si por separado iban bien? ¡Quién sabe! Lo que es seguro es que muchos usuarios estarían deseando ver el código interno de *Mathematica* para averiguar a qué se debe tan llamativo fallo y poder repararlo. Lamentablemente, es imposible, pues sólo Wolfram Research tiene acceso a él. Abundando más en esto, diremos que este error no estaba presente en la versión 3.0 de *Mathematica*, sino que apareció con la versión 4, y ha sido divulgado en los foros de internet adecuados (aunque no tal como aquí lo hemos presentado); siendo así, ¿por qué no lo corrigen los únicos que pueden hacerlo?

5 Tratamiento de casos

En las secciones anteriores hemos comentado, en varias ocasiones, que *Mathematica* acostumbra a ser muy cuidadoso cuando tiene que simplificar. Misteriosamente, otras veces parece pasar por alto toda precaución. Todos los

[16] Para valores impares de la variable, es muy poco lo que se sabe: está demostrado que $\zeta(3)$ es irracional, pero se desconoce si es o no un número algebraico; de $\zeta(5)$, $\zeta(7)$, etc., ni siquiera se sabe si son o no números racionales. La función $\zeta(z)$ se denomina zeta de Riemann, y adquiere toda su importancia cuando se extiende al campo complejo. Sorprendentemente, tener información sobre las raíces complejas de $\zeta(z)$ resulta ser fundamental en teoría de números. En particular, se usa para probar resultados sobre la distribución de números primos. La hipótesis de Riemann afirma que todos los ceros ≪no triviales≫ de $\zeta(z)$ se encuentran sobre esta recta del plano complejo: $z = \frac{1}{2} + ti$, $t \in (-\infty, \infty)$. De momento, esto no deja de ser una conjetura. Hay un premio de un millón de dólares esperando a quien la demuestre o la refute.

que están habituados a tratar con sistemas ortogonales –y a cualquier otro matemático le resultará muy sencillo comprobarlo– saben que, cuando n y m son enteros,

$$\int_0^{\pi} \cos(nt)\cos(mt)\,dt = \begin{cases} 0, & \text{si } n \neq m, \\ \pi/2, & \text{si } n = m \neq 0, \\ \pi, & \text{si } n = m = 0. \end{cases} \tag{1}$$

Sin embargo, *Mathematica* yerra clamorosamente al calcular esta integral. Las respuestas concretas dependen de la versión que estemos usando pero, en esencia, podemos afirmar que *Mathematica* no se da cuenta de que n y m pueden ser iguales; y si evaluamos $\int_0^{\pi} \cos^2(nt)\,dt$, pasa por alto la posibilidad $n = 0$. Ninguna versión calcula correctamente la integral propuesta en (1). Así, por ejemplo, *Mathematica* 5.0 y 5.1 evalúan

```
Integrate[Cos[n*t]*Cos[m*t], {t, 0, Pi},
  Assumptions -> {Element[{n, m}, Integers]}]
```

como 0; y si hacemos la correspondiente integral con $n = m$, obtenemos $\pi/2$. (Las versiones anteriores proporcionan respuestas más oscuras, pero también equivocadas.)

Mathematica tampoco anda muy fino cuando lo enfrentamos a algo tan cotidiano como calcular las soluciones de un sistema lineal. La orden `Solve`, destinada a resolver ecuaciones o sistemas, no tiene en cuenta los posibles valores de los parámetros. Así, ante el sistema lineal

```
Solve[{2x + a*y == 3, 4x + 8y == 6}, {x, y}]
```

sólo nos proporciona la solución $x = 3/2$, $y = 0$; no se percata de que, si $a = 4$, hay muchas más soluciones. Si le proponemos un sistema que, para algunos valores del parámetro, es incompatible, tampoco lo nota. Es más, todos estos fallos persisten si intentamos resolver estos sistemas por medio del comando `LinearSolve`, que es específico para sistemas lineales. En realidad, se puede lograr que *Mathematica* analice adecuadamente estos problemas si, en lugar de `Solve`, usamos `Reduce`; pero, ¿quién se lo podía esperar? No parece razonable que los creadores de *Mathematica* esperen que el usuario se haya estudiado concienzudamente su voluminoso manual [15] antes de manejarlo.

6 Una pifia espectacular

Algunas veces, el patinazo es morrocotudo. *Mathematica* 5.0 afirma que

```
Integrate[(1 - Exp[-t])^2 * t^(-3/2), {t, 0, Infinity}]
```

vale $-2\sqrt{2\pi}$. Esto es claramente imposible, ya que el integrando es positivo; es más, si no lo hubiera sido, difícilmente hubiéramos sospechado que la respuesta es errónea, pues nada parece indicar que la integral tenga alguna complicación oculta (obsérvese que $\lim_{t\to 0}(1-e^{-t})^2 t^{-3/2} = 0$). Sí que podíamos habernos dado cuenta comparando con el valor aproximado de la integral calculado mediante un procedimiento numérico (lo que se consigue con `NIntegrate`), pues esto sí proporciona una respuesta satisfactoria.

Afortunadamente, parece que sólo ha sido un problema momentáneo. El resto de las versiones de *Mathematica* que hemos evaluado, tanto anteriores como posteriores, proporcionan el valor correcto

$$\int_0^\infty (1-e^{-t})^2 t^{-3/2}\,dt = (4-2\sqrt{2})\sqrt{\pi}.$$

Que la versión 5.1 haya arreglado el problema de la 5.0 es bueno, pero no nos hace muy felices. Es evidente que, cuando aparece una nueva versión de un programa, contiene rutinas nuevas; en lo que ahora nos concierne, las dedicadas a la integración. Aunque sin dar detalles, esto lo suelen indicar los creadores de *Mathematica*, pues –lógicamente– les gusta destacar las mejoras que introducen. Pero, ¿por qué no nos señalan igualmente los fallos que han corregido? Sólo podemos pensar una cosa: nos quieren hacer creer que sus programas no tienen fallos, inducirnos a confiar ciegamente en ellos. Este oscurantismo es, desde todo punto de vista, lamentable.

Analicemos ahora otro ejemplo; esta vez –todo hay que decirlo– no tan grave. De nuevo tiene como protagonista principal a *Mathematica* 5.0, que afirma que la integral

$$\int_0^\infty \frac{\log(t)}{\sqrt{t}(1+t)}\,dt \tag{2}$$

no converge en $(0,\infty)$. Bastan unos instantes de reflexión para convencernos de que eso no es cierto. Además, resulta sencillo calcular su valor; para ello, sin más que efectuar el cambio $t = 1/s$, es inmediato comprobar que

$$\int_0^1 \frac{\log(t)}{\sqrt{t}(1+t)}\,dt = -\int_1^\infty \frac{\log(s)}{\sqrt{s}(1+s)}\,ds,$$

de donde se sigue que (2) vale 0. Sorprendentemente, *Mathematica* 5.0 sí que sabe calcular la integral sobre $(0,1)$, y nos indica –correctamente– que su valor es `-4Catalan` (como el lector habrá podido adivinar, `Catalan` alude a cierta constante, cuya definición concreta y valor aproximado no tienen ahora importancia; pero, para satisfacer su curiosidad, aclaremos que la denominada constante de Catalan es $\sum_{n=0}^\infty (-1)^n (2n+1)^{-2} = 0.915966\ldots$). *Mathematica* 5.0 sí que logra calcular (2) si le indicamos que «preste atención» al punto $t = 1$, lo que se consigue con

```
Integrate[Log[t]/(Sqrt[t]*(1 + t)), {t, 0, 1, Infinity}]
```

En realidad, esto está pensado para los puntos singulares, y $t = 1$ no lo es, pero lo reseñamos pues esta misma idea puede ser de ayuda en otras ocasiones. Hasta ahora hemos relatado lo que hace la versión 5.0; ¿cómo se comportan otras versiones cuando las enfrentamos con (2)? Las anteriores calculan esa integral sin problemas, y con rapidez. También la versión 5.1 sabe evaluar (2) pero, misteriosamente, emplea mucho tiempo (medio minuto, en un ordenador fabricado en 2004). Esto nos hace pensar: ¿por qué, si los programadores han detectado un fallo que antes no estaba, no lo han arreglado hasta dejarlo tal como funcionaba cuando lo hacía bien? ¿Quizás lo han corregido por casualidad, sin darse cuenta? Misterios.

7 Despistes y complicaciones innecesarias al integrar

En general, si prescindimos de fallos como los que acabamos de relatar, *Mathematica* es muy potente en cuanto a integración simbólica se refiere. Podemos dar por sentado que es mejor que cualquier matemático experto, y que muchos libros de tablas de integración. Los ejemplos de la sección anterior hacían todos referencia a integrales definidas; vamos ahora a dar algunas muestras de su comportamiento frente a integrales indefinidas –cálculo de primitivas–. Antes de seguir, conviene que reflexionemos sobre la dificultad e imprecisión de esta tarea, siendo éste otro tema que preocupa a los teóricos del cálculo simbólico (véase [8]). Hay que darse cuenta de que no siempre está bien planteado el significado de $\int f(x)\,dx$ (`Integrate[f[x], x]`, con la sintaxis de *Mathematica*). Siendo rigurosos, deberíamos aclarar en qué rango de valores de la variable x pretendemos que sea válido el resultado. Dependiendo del dominio, serán unos u otros los cambios de variable que se puedan efectuar, por ejemplo. Usualmente, ni siquiera nos preocupamos por ello cuando calculamos integrales «a mano»: lo que obtenemos tendrá sentido en algún dominio, que seríamos capaces de precisar si fuera necesario, pero no solemos hacerlo. Además, diversas estrategias de abordar el problema pueden dar lugar a expresiones aparentemente muy distintas. Por último, tampoco se especifica en qué términos se desea expresar la integral: es obvio que $\int_a^x f(t)\,dt$ es una respuesta cierta, pero no muy útil. Afortunadamente, ni que decir tiene que el método para detectar posibles errores es ahora relativamente sencillo: ¡basta derivar lo que hemos obtenido al integrar!

Aunque no podemos saber exactamente cuál es su funcionamiento interno, lo razonable es pensar que *Mathematica* incorpora una amplia tabla de integrales conocidas, y mecanismos de sustitución, cambio de variable, etc., para reducir otras integrales que le propongamos a las de la tabla.

Hay veces en las que, inexplicablemente, no logra hacerlo. Por ejemplo, todas las versiones de *Mathematica* que hemos usado (la última, recordemos,

la 5.1) saben calcular

$$\int \frac{x^2 e^{\operatorname{arctg}(x)}}{\sqrt{1+x^2}}\,dx = \frac{e^{\operatorname{arctg}(x)}(-1+x-x^2+x^3)}{2\sqrt{1+x^2}};$$

sin embargo, devuelven

$$\int \frac{x^2 e^{-\operatorname{arctg}(x)}}{\sqrt{1+x^2}}\,dx$$

sin evaluar. *Mathematica* no se da cuenta de que, con el cambio de variable $x=-t$, esta integral se transforma en la anterior. Bueno, no saber hacer algo no es una respuesta errónea. ¡Pero parece mentira!, con tanta potencia y tanto algoritmo interno.

En algunas ocasiones, y aunque *Mathematica* proporciona una respuesta correcta, lo hace por medio de una expresión más complicada que la que podría obtener –sin dificultad y de manera directa– cualquier matemático. Veamos algunos ejemplos. Esta vez, las respuestas no van a ser unánimes, sino que dependerán de la versión de *Mathematica* que estemos usando. Resulta curioso observar que, unas veces, son las versiones más antiguas las que proporcionan respuestas más naturales; otras, al contrario.

Para calcular

$$\int \frac{\operatorname{sen}(x)}{\sqrt{2}\cos(x)-\sqrt{\cos(x)}}\,dx,$$

en el denominador sacamos $\sqrt{\cos(x)}$ factor común, y efectuamos el cambio $u=\sqrt{2\cos(x)}-1$. Así,

$$\int \frac{\operatorname{sen}(x)\,dx}{\sqrt{2}\cos(x)-\sqrt{\cos(x)}} = -\sqrt{2}\int \frac{du}{u} = -\sqrt{2}\log\left(-1+\sqrt{2}\sqrt{\cos(x)}\right).$$

Mathematica 5.0 y 5.1 obtienen directamente este resultado, pero las versiones anteriores nos daban una primitiva mucho más enrevesada:

$$\int \frac{\operatorname{sen}(x)\,dx}{\sqrt{2}\cos(x)-\sqrt{\cos(x)}} = -\frac{\sqrt{2}\big(-1+\sqrt{2}\sqrt{\cos(x)}\big)\sqrt{\cos(x)}\log(-1+\sqrt{2}\sqrt{\cos(x)})}{-\sqrt{\cos(x)}+\sqrt{2}\cos(x)}.$$

A decir verdad, si empleamos `Simplify` también llegamos a la expresión sencilla. En todo caso, esto es esperanzador; parece que las nuevas versiones son un avance.

Pero veamos qué ocurre con otra integral trigonométrica:

$$\int \frac{\operatorname{sen}(x)\,dx}{\sqrt{2\cos(x)-(\cos(x))^2}}.$$

El radicando del denominador se puede escribir como $1-(1-\cos(x))^2$, y entonces hacer el cambio $u=1-\cos(x)$. De este modo, es claro que

$$\int \frac{\operatorname{sen}(x)\,dx}{\sqrt{2\cos(x)-(\cos(x))^2}} = \int \frac{du}{\sqrt{1-u^2}} = \operatorname{arcsen}(u) = \operatorname{arcsen}(1-\cos(x)).$$

Esta vez, son *Mathematica* 3.0, 4.1 y 4.2 los que proporcionan directamente la primitiva $\operatorname{arcsen}(1-\cos(x))$. Sin embargo, *Mathematica* 5.0 y 5.1 evalúan la integral como

$$\int \frac{\operatorname{sen}(x)\,dx}{\sqrt{2\cos(x)-(\cos(x))^2}} = -\frac{2\sqrt{-2+\cos(x)}\sqrt{\cos(x)}\log(\sqrt{-2+\cos(x)}+\sqrt{\cos(x)})}{\sqrt{-(-2+\cos(x))\cos(x)}}. \qquad (3)$$

En este ejemplo, simplificar tampoco ayuda; no hemos logrado recuperar la respuesta $\operatorname{arcsen}(1-\cos(x))$ que hemos calculado nosotros mismos. Así pues, no da la impresión de que las nuevas versiones sean siempre mejores, como sería deseable. Alguien podría argumentar que la primitiva (3) no es tan mala; que, simplemente, estamos hallando otra expresión, y que nunca está claro qué se quiere decir con «simplificar». Quien piense de manera tan benévola es que no se ha fijado en la fórmula que nos han proporcionado *Mathematica* 5.0 y 5.1: en ella aparece varias veces $\sqrt{-2+\cos(x)}$, ¡un radicando que siempre es negativo!

Somos matemáticos, y no hemos podido resistirnos a analizar un poco más qué estaba pasando. ¿Será la respuesta (3) correcta si la desarrollamos como números complejos?; ¿o quizás es, simplemente, un error? Siempre cabe la posibilidad, incluso, de que las varias expresiones complejas que, forzosamente, aparecen en (3), se cancelen unas con otras. La realidad es una mezcla de todo esto. Hemos logrado comprobar que, en un intervalo real centrado en $x=0$, la función (3) se diferencia de $\operatorname{arcsen}(1-\cos(x))$ en $2i\log(1+i)$, siendo $i=\sqrt{-1}$. Así pues, rigurosamente hablando, la primitiva hallada por *Mathematica* 5.0 y 5.1 no es errónea, ya que se diferencia en una constante (¡aunque compleja!) de la hallada por nosotros (y por las versiones anteriores de *Mathematica*). Pero nadie podrá negar que $\operatorname{arcsen}(1-\cos(x))$ es una primitiva mucho más apropiada.

8 Tres ejemplos numéricos llamativos

En algunas ocasiones, *Mathematica* dispone de dos comandos similares para realizar la misma tarea; uno para hacerla de manera simbólica y otro de forma numérica. Así, por ejemplo, existen los comandos `Integrate` y `NIntegrate`, `Sum` y `NSum`, `Solve` y `NSolve`. En las «órdenes simbólicas» (entre las anteriores, las tres que no comienzan por `N`), si la expresión que estamos evaluando no contiene números decimales, *Mathematica* efectúa las operaciones

de manera exacta (por ejemplo, muestra fracciones o raíces en lugar de su correspondiente valor numérico aproximado); en caso contrario, lleva a cabo aproximaciones con un determinado número de decimales (pero no acude al correspondiente comando con `N`). Este tipo de órdenes puede dar lugar a problemas inesperados, como mostramos a continuación. Veremos que unos métodos son más apropiados que otros; pero, ¿cómo saberlo previamente si no tenemos suficiente dominio de la cuestión?

En primer lugar, observemos que, cuando t es negativo, $e^t = \sum_{k=0}^{\infty} t^k/k!$ es una serie alternada. El teorema de Leibniz asegura que el error que se comete al aproximar e^t por $\sum_{k=0}^{n} t^k/k!$ está acotado por $|t|^{n+1}/(n+1)!$. Así, si tomamos $t = -40$ y $n = 300$, debe ser

$$\sum_{k=0}^{300} \frac{t^k}{k!} \approx e^{-40} \approx 4.24835 \cdot 10^{-18},$$

con un error realmente despreciable (mucho menor que las cifras significativas que hemos mostrado). ¿Qué hace *Mathematica*? Pues depende de cómo lo usemos. Imaginemos que estamos intentado obtener el valor numérico de esa suma y que, ingenuamente, escribimos

```
t = -40.0; Sum[t^k/k!, {k, 0, 300}]
```

Al ejecutarlo, *Mathematica* nos proporciona una respuesta totalmente equivocada. El valor concreto que nos da puede depender de la versión que estemos usando;[17] por ejemplo, *Mathematica* 5.0 y 5.1 calculan esa suma como -3.52833, que no se parece al valor $4.24835 \cdot 10^{-18}$ que debería haber tenido. Podemos adivinar lo que ha pasado. En $\sum_{k=0}^{300}(-40)^k/k!$, los primeros sumandos son, en valor absoluto, bastante grandes (luego, el denominador crece mucho más rápido que el numerador, y por eso la serie converge). El número de cifras significativas que está manejando *Mathematica* es inferior al que necesitaría almacenar para que todas las cancelaciones entre términos de distinto signo no se perdieran. Eso hace que acumule un error que no es significativo frente al tamaño de los sumandos grandes, pero sí frente al resultado final, que es casi cero.

Como entendemos el funcionamiento de la máquina, logramos comprender qué está pasando, e incluso podíamos haber previsto que éste no era un buen método de cálculo. Mucho más provechoso hubiera sido haber tomado $t = -40$ (exacto, sin ser un número decimal), haber efectuado la suma (que hubiera salido una fracción enorme), y luego tomado su aproximación decimal, así:

[17] Como se trata de un proceso puramente numérico, que posiblemente descanse sobre el hardware y las rutinas de cálculo de la máquina de una manera bastante directa, no sería de extrañar que también dependiera del microprocesador y del sistema operativo del ordenador que hayamos empleado. Pero, a decir verdad, esto no lo hemos observado.

```
t = -40; Sum[t^k/k!, {k, 0, 300}] // N
```

Eso ya proporciona sin problemas el valor $4.24835 \cdot 10^{-18}$. Pero –y perdónenos el lector por nuestra insistencia– esto requiere tener claro qué es lo que estamos haciendo; eso sólo se consigue con conocimientos matemáticos –en particular, de cálculo numérico y su interacción con los procesos internos de los ordenadores–. Como moraleja, conviene recordar que, cuando se pueda, suele ser conveniente operar de manera exacta, y al final hacer aproximaciones numéricas. Aunque, como a veces el fallo está en alguna rutina simbólica (¡ya hemos visto algunos ejemplos!), no está de más intentar siempre ambos caminos.

La orden `NSum` está especializada en llevar a cabo procesos numéricos. Así, uno podría esperar que, internamente, *Mathematica* tomara precauciones y que, como consecuencia,

```
t = -40.0; NSum[t^k/k!, {k, 0, 300}]
```

se iba a comportar mejor. Algo mejor sí que lo hace: se da cuenta de que quizás la suma no la ha calculado bien y, antes de proporcionarnos la supuesta suma, nos da un aviso que indica que el resultado puede ser incorrecto. Tras la advertencia, *Mathematica* 3.0 aventura que la suma vale 4261.26. Por el contrario, *Mathematica* 4.1 y 4.2 continúan calculando durante bastante rato; al final, nos cansamos de esperar y abortamos el proceso. *Mathematica* 5.0 y 5.1 también se atreven a dar un resultado, aunque disparatado: $4261.32 - 9.61518 \cdot 10^{-149} i$, con $i = \sqrt{-1}$. Lo mismo ocurre si, con `NSum`, ponemos `t = -40` en lugar de `t = -40.0`.

Vamos ahora a ver un ejemplo similar, pero con integrales en vez de con series. Como e^{-x^2} es despreciable para valores grandes de $|x|$, se tiene

$$\int_{-1000}^{1000} e^{-x^2}\, dx \approx \int_{-\infty}^{\infty} e^{-x^2}\, dx = \sqrt{\pi} \approx 1.77245.$$

Si intentamos evaluar numéricamente la expresión de la izquierda mediante

```
NIntegrate[Exp[-x^2], {x, -1000.0, 1000.0}]
```

de nuevo obtenemos un aviso, y luego aproximaciones desafortunadas. Tampoco es siempre satisfactorio lo que conseguimos si, manteniendo `-1000.0` y `1000.0`, usamos `Integrate` en vez de `NIntegrate`. Esta vez, hay más variación de resultados dependiendo de la versión usada. Comentemos algunos de ellos:

- Con `NIntegrate`, *Mathematica* 3.0, 4.1, 4.2 y 5.0 afirman que la integral vale $1.349346 \cdot 10^{-26}$. Eso sí, con la correspondiente advertencia de que posiblemente esté mal evaluada.

- Usando `Integrate`, *Mathematica* 5.0 calcula la integral correctamente.

- Tanto con `Integrate` como con `NIntegrate`, *Mathematica* 5.1 efectúa los cálculos de forma impecable (¡bravo por él!; esta vez ha sido el mejor).

- Resulta curioso observar que, con `Integrate`, *Mathematica* 3.0 es capaz de evaluar la integral; pero para ello emplea alrededor de cinco minutos (medido en un ordenador no muy moderno), y muestra el resultado con cientos de miles de decimales. *Mathematica* 4.1 hace lo mismo (y tarda unos 20 segundos en un ordenador de 2004). Tras una larga espera, no hemos tenido paciencia para ver a qué llegaba *Mathematica* 4.2, y hemos abortado los cálculos.

Como ocurría con las series, lo recomendable es hacer

```
Integrate[Exp[-x^2], {x, -1000, 1000}] // N
```

Pero no siempre es factible recurrir a esto. Si nos hubiéramos topado con una función que *Mathematica* no sabe cómo tratar de manera exacta, este método no hubiera servido de nada.

Una chapuza gorda de *Mathematica* 5.0 se pone de manifiesto ante la integral compleja

```
NIntegrate[1/(1+z^2), {z, 0, I}]
```

(recordemos que `I` denota la unidad imaginaria). Es claro que es divergente, pues en $z = \sqrt{-1}$ hay un polo. Y de ello se dan cuenta todas las versiones de *Mathematica* si hacemos la integral de manera simbólica (con `Integrate`). Al efectuarla numéricamente, casi todas las versiones se comportan de manera aceptable: proporcionan un valor –erróneo–, pero previamente advierten de que dudan de la respuesta. Salvo *Mathematica* 5.0, que sin pestañear afirma que esa integral vale $0.0 + 1.77949i$.

Por el contrario, no pensemos que, en cuanto a integración se refiere, *Mathematica* 5.0 sólo sirvió para empeorar cosas que funcionaban bien. Esta vez, veamos algo que ha arreglado. Las versiones de *Mathematica* hasta la 4.2 incluida afirmaban que

```
Integrate[ArcSec[z], {z, 0, 1}]
```

vale 0. Pero, al calcular esa integral con `NIntegrate`, encontraban el valor $1.5708i$ (obsérvese que $\sec(x) = 1/\cos(x) \notin (-1, 1)$ cuando x es un número real, luego el integrando de la expresión anterior no es una función real, sino compleja). *Mathematica* 5.0 y 5.1 sí que calculan la integral simbólica correctamente, y nos dan el valor exacto, $i\pi/2$, que se ajusta al obtenido numéricamente.

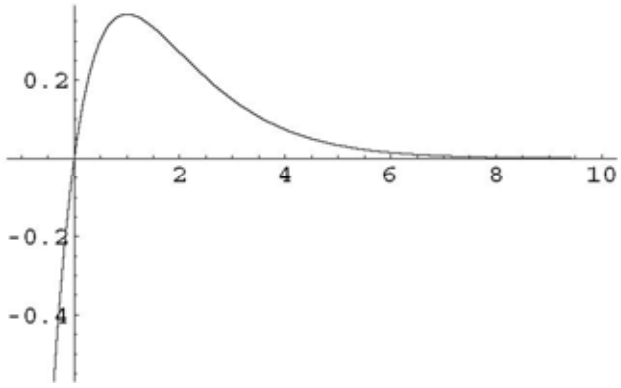


Figura 4: Gráfica de la función $y = xe^{-x}$.

9 Cálculos numéricos inesperados

Hay veces que no está claro si la orden que estamos usando va a dar lugar a un comportamiento simbólico o numérico. Quizás se pueda considerar un fallo de diseño que esto no esté mejor explicitado (aunque siempre se pueden consultar las instrucciones, por supuesto).

Algunas veces, esto da lugar a situaciones que –simplemente– se pueden considerar un poco ridículas. Por ejemplo, imaginemos que queremos aproximar la tabla de datos `{{0, 0}, {1, 1}, {2, 4}}` por medio de una combinación lineal de las funciones 1, x y x^2. Esto se hace mediante la orden

```
Fit[{{0, 0}, {1, 1}, {2, 4}}, {1, x, x^2}, x]
```

Es obvio que, sea cual sea el tipo de aproximación que se pretenda efectuar –si leemos el manual o la ayuda del programa nos enteramos de que `Fit` efectúa aproximación por mínimos cuadrados–, la respuesta lógica es que la función que mejor lo hace es x^2. Sin embargo, *Mathematica* no nos proporciona esa función, sino algo similar a $2.52912 \cdot 10^{-16} + 3.97205 \cdot 10^{-16} x + 1.0x^2$ (ésa es la respuesta concreta de *Mathematica* 5.0 y 5.1; otras versiones nos dan coeficientes ligeramente diferentes). ¿Qué es lo que ha sucedido? Es fácil adivinarlo: *Mathematica* ha hecho un tratamiento numérico del problema y, como es habitual en todo proceso numérico, ha introducido pequeños errores de redondeo. La verdad es que no tiene demasiada importancia práctica (y, además, *Mathematica* dispone de la orden `Chop` para librarse de esos «presuntos ceros» en los coeficientes), pero no podemos negar que la situación resulta graciosa.

En otras ocasiones, los métodos numéricos pueden fallar por completo. Es claro que la función xe^{-x} sólo tiene una raíz: $x = 0$ (véase, además, la

figura 4). Sin embargo, imaginemos que queremos encontrarla mediante el comando `FindRoot`. Tras leer las instrucciones, nos enteramos de que `FindRoot` utiliza un proceso iterativo para resolver ecuaciones, y que hay que proporcionarle un punto inicial (aunque escuetamente, esta vez *Mathematica* nos suministra algo de información de lo que va a hacer internamente: nos dice que «utiliza los métodos de Newton, de la secante o de Brent»). Si ejecutamos la orden

```
FindRoot[x*Exp[-x], {x, 3}]
```

tanto *Mathematica* 3.0 como 4.1 y 4.2 nos proporcionan la supuesta raíz $x = 16.9273$. A nuestro entender, podemos considerar que esto es un fallo garrafal. Posiblemente se haya producido porque no se ha utilizado una condición de parada adecuada. Ante un método iterativo que, para intentar resolver $f(x) = 0$, genera una serie de valores $\{x_n\}_{n=0}^{\infty}$, hay dos tipos de condiciones que se suelen usar: comprobar que, para ciertos valores de $\varepsilon, \varepsilon'$ suficientemente pequeños, se verifica $|f(x_n)| < \varepsilon$ (es decir, x_n es una raíz aproximada) y $|x_{n+1} - x_n| < \varepsilon'$ (x_n es, aproximadamente, un punto fijo del método que busca raíces). Si, sobre la figura 4, nos imaginamos cuál va a ser el comportamiento del proceso iterativo, los valores de x_n que parten de $x_0 = 3$ van a ir creciendo, luego el método no va a poder encontrar ninguna raíz verdadera. Parece que *Mathematica* sólo ha comprobado si $|f(x_n)|$ es muy pequeño; esto seguro que ocurre alguna vez, pues xe^{-x} decrece muy rápido cuando $x \geq 3$. Es verdad que $x = 3$ es un mal punto de partida para encontrar una raíz de $xe^{-x} = 0$; pero eso no vale como excusa: ¿por qué –quizás en otro ejemplo menos obvio– el usuario debería saberlo? Puede ser razonable que el programa no encuentre una raíz, pero no que nos engañe.

Mathematica 5.0 y 5.1 tampoco hallan la raíz, e incluso aventuran una solución más alejada de la verdadera: $x = 106.795$. Aunque hay que reconocer que, esta vez, su comportamiento es aceptable, pues previamente avisan de que no han podido comprobar la convergencia del método.

10 Raíces de polinomios

Hace años, nos encontrábamos inmersos en una investigación teórica sobre ciertas propiedades asintóticas de los ceros de polinomios ortogonales sobre la circunferencia unidad del plano complejo (véase [2]). Habíamos demostrado un nuevo teorema, y queríamos ilustrarlo gráficamente con algunos ejemplos; para ello necesitábamos construir ciertos polinomios de grado alto (alrededor de 100, por ejemplo), y hallar sus raíces. En ese momento, *Mathematica* iba por la versión 3.0. Allí hicimos nuestros dibujos, que se ajustaban sin problemas a lo que nos decían los resultados teóricos. Es más; previamente, *Mathematica* nos había sido muy útil como herramienta experimental. Las numerosas variaciones que habíamos ensayado nos habían llevado a conjeturar cuál tenía que ser el

resultado teórico que perseguíamos, para así saber qué teníamos que intentar probar.

Como todos los que nos dedicamos a la investigación matemática sabemos, los procesos de publicación en esta ciencia no son tan rápidos como en otras. Esto se juntó con otros avatares que no merece la pena recordar. El caso es que, entre la primera versión del artículo, la inclusión de los cambios sugeridos por los *referees*, y su aceptación definitiva para ser publicado, pasaron unos cuantos años. En ese tiempo, *Mathematica* había cambiado varias veces de versión. Es más, incluso los ordenadores que habíamos usado para hacer las gráficas estaban ya obsoletos, y teníamos otros mucho más rápidos.

Quisimos hacer alguna modificación sin importancia en las gráficas (por ejemplo, eliminar el color y cambiarlo por tonos de gris, pues las revistas matemáticas no suelen publicar en colores). El caso es que, tras efectuar estas pequeñas modificaciones, y conservando toda la parte esencial del código en *Mathematica* que habíamos usado previamente, repetimos los cálculos con los nuevos ordenadores y sus versiones actualizadas de *Mathematica*. ¡Qué desastre! El código se ejecutaba sin problemas aparentes, pero los dibujos que obteníamos eran muy distintos. Ya no eran útiles para ilustrar los teoremas. ¿Qué estaba pasando?; ¿qué funcionaba mal? Quizás nuestros ejemplos no servían para nada; cuando nos topamos por primera vez con este problema éramos más ingenuos que ahora, y pensábamos que las respuestas proporcionadas por las versiones nuevas debían ser las más ajustadas a la realidad.

Nuestros programas tenían cierta componente numérica, quizás propensa a errores –tal como ya hemos comentado varias veces, desconocemos qué algoritmos ha usado *Mathematica* internamente, luego no podemos saber cuál va a ser su comportamiento–. Sin embargo, al ejecutarlos con *Mathematica* 3.0, producían resultados razonables, y eran acordes con lo que los teoremas que estábamos enunciando preveían. Nada hacía pensar que lo que obteníamos pudiera ser ficticio.

El caso es que nosotros estábamos dibujando configuraciones de ceros de polinomios ortogonales sobre la circunferencia unidad. Una importante propiedad de estos ceros es que siempre deben estar dentro de la circunferencia. En seguida nos dimos cuenta de que lo que estaban haciendo las versiones nuevas de *Mathematica* tenía que ser forzosamente erróneo, pues estaban encontrando supuestas raíces en el exterior de la circunferencia. Nos parecía mentira, pero algo que funcionaba bien, ahora ya no lo hacía. Eso sí, todo era mucho más rápido; no sólo por el cambio de ordenador, sino que una gran parte de la mejora de velocidad era manifiestamente imputable a los algoritmos que estaba usando *Mathematica*. Cada vez estaba más claro. Las nuevas versiones de *Mathematica* anunciaban espectaculares mejores de rendimiento. Lo que no advertían es que, al parecer, lo habían logrado a costa de utilizar algoritmos mucho más inestables.

No vamos a repetir ahora los cálculos que hacíamos entonces, y que tenían cierta sofisticación que no tiene aquí ningún interés. En su lugar, hemos buscado cómo aislar el problema, mostrando el tipo de comportamiento que hemos

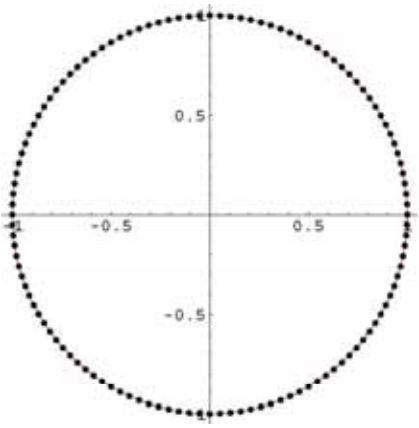


Figura 5: Las raíces de $z^{120} - 1 = 0$, según *Mathematica* 3.0 (y 4.1).

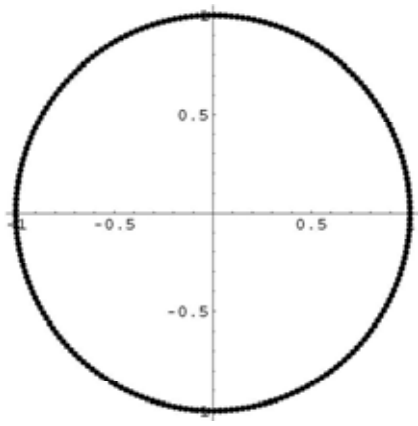


Figura 6: Las raíces de $z^{217} - 1 = 0$, según *Mathematica* 3.0 (y 4.1).

relatado en un ejemplo sencillo. Lo hemos logrado de una manera que –quizás– puede resultar incluso sorprendente: vamos a pedir a *Mathematica* que, para diversos valores concretos de n, halle todas las raíces del polinomio $z^n - 1 = 0$. Sabemos que las raíces son $e^{2k\pi i/n} = \cos(2k\pi/n) + i\text{sen}(2k\pi/n)$, $k = 0, 1, \ldots, n-1$, pero le vamos a proponer a *Mathematica* que las calcule mediante un algoritmo numérico. Obviamente, esto podía haber sido resuelto de manera simbólica; pero no hubiera sido así con polinomios más complicados, como ocurría con los que teníamos cuando nos topamos con el problema. Además, nuestro principal interés radica ahora en ver si las nuevas versiones lo hacen mejor o peor (reiteramos que los algoritmos numéricos presentan muchas dificultades, así que quizás podríamos haber aceptado que, para n grande, *Mathematica* nunca hubiera sido capaz de encontrar las raíces).

Así pues, veamos cómo calculan las raíces de $z^n - 1$ las diversas versiones de *Mathematica* que estamos analizando, siempre de manera numérica. Para ello, tras asignar un valor a n, le indicamos a *Mathematica* que ejecute la orden

```
NSolve[z^n - 1 == 0, z]
```

Tras ello, representamos las raíces (en el plano complejo) y la circunferencia unidad. Si el cálculo estuviera bien hecho, deberíamos obtener n puntos ($e^{2k\pi i/n}$, con $k = 0, 1, \ldots, n-1$) regularmente distribuidos sobre la circunferencia. Hemos hecho bastantes experimentos, con n tomando valores hasta 500. Hay tanta casuística que es difícil resumir las respuestas más significativos que aparecen, pero vamos a intentarlo.

Con *Mathematica* 3.0 siempre encontramos resultados aceptables, del estilo que podemos ver en las figuras 5 y 6. Y los tiempos de cálculo son siempre breves. Al menos en lo que nosotros hemos observado, *Mathematica* 4.1 halla las mismas gráficas, pero para ello emplea un tiempo decenas de veces mayor.

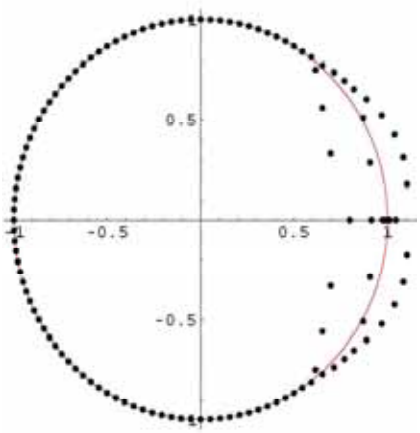


Figura 7: Las raíces de $z^{120} - 1 = 0$, según *Mathematica* 4.2.

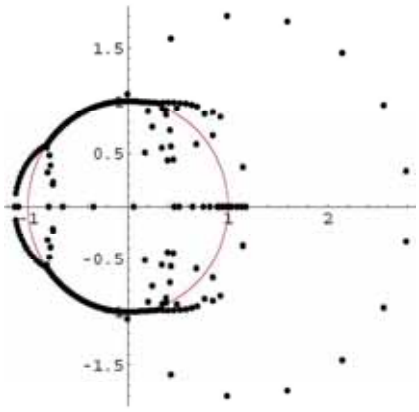


Figura 8: Las raíces de $z^{217} - 1 = 0$, según *Mathematica* 4.2.

A partir de $n = 75$, *Mathematica* 4.2 comienza a dar muestras de pérdida de precisión al hallar las raíces. Así, por ejemplo, representa las raíces de $z^{120} - 1 = 0$ como en la figura 7, y las de $z^{217} - 1 = 0$ como en la figura 8; eso sí, es muy rápido. Misteriosamente, y según vamos aumentando el valor de n, de repente nos topamos con algunos n para los que *Mathematica* emplea mucho más tiempo de cálculo del que nos tenía acostumbrado (típicamente, 100 veces mayor), y nos proporciona un dibujo impecable. Así ocurre, por ejemplo con $n = 219$ (pese a que 218 y 220 sí que daban problemas). Éste también es el caso con $n = 250$, 400 y 500; pero el gráfico correspondiente a 300 es erróneo (y lo obtiene muy rápidamente). Da la impresión de que *Mathematica* está usando, al menos, dos algoritmos diferentes; uno rápido pero inestable, y otro lento y preciso; desconocemos el motivo que le induce a emplear uno u otro.

Al principio, *Mathematica* 5.0 parece comportarse de manera similar a como lo hacía la versión 4.2; aunque, cuando falla al encontrar las raíces, no obtiene exactamente lo mismo. Pero, a partir de $n = 219$, parece que siempre obtiene dibujos acordes a lo que se esperaba (al menos, en los experimentos que hemos hecho no hemos encontrado ninguno incorrecto); por ejemplo, dibuja bien los gráficos correspondientes a $n = 220$ y 300, con los que *Mathematica* 4.2 tenía problemas.

Mathematica 5.1 de nuevo empeora las cosas. Sus dificultades también comienzan a observarse con valores de n próximos a 75. A partir de ahí, muchos de los dibujos que obtiene son similares a los de las versiones 4.2 y 5.0; aunque, al contrario que estas, logra dibujar correctamente el correspondiente a $n = 200$. Así hasta llegar al enigmático $n = 219$ que, como es habitual, y tras tomarse su tiempo, representa correctamente. Luego, parece no volver a acertar. Como ejemplo, mostramos lo que obtiene en relación con $n = 400$ (figura

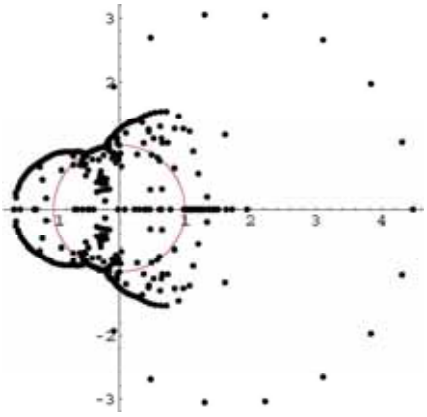


Figura 9: Las raíces de $z^{400} - 1 = 0$, según *Mathematica* 5.1.

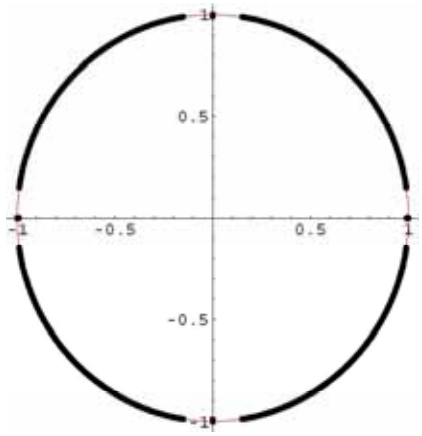


Figura 10: Las raíces de $z^{500} - 1 = 0$, según *Mathematica* 5.1.

9) y 500 (figura 10); ambos eran tratados correctamente por *Mathematica* 4.2 y 5.0 (y también por las versiones 3.0 y 4.1). Vale la pena comentar que, por el tiempo empleado, parece que con $n = 500$ ha usado el supuesto «algoritmo lento»; y que el tipo de dibujo que se obtiene –en el que no aparecen los ceros cercanos a 1, i, -1 y $-i$ (o están concentrados justo sobre esos puntos, a simple vista no se puede asegurar qué ocurre)– no lo hemos observado en ninguna otra versión de *Mathematica*.

Curioso, ¿no? Concluyamos la sección comentando que en los ejemplos que estábamos usando en [2] aún obteníamos más variedad de resultados, que no hemos logrado reproducir con polinomios de tipo $z^n - 1$:

- Aparecían ejemplos en los que *Mathematica* 3.0 no tenía dificultades, pero *Mathematica* 4.1 fallaba.
- A veces, *Mathematica* 4.1 nos daba mensajes de error incomprensibles, y encontraba muchos menos ceros que el grado del polinomio.
- Surgían circunstancias bajo las que *Mathematica* 5.0 se comportaba peor que 5.1, algo que no hemos observado con $z^n - 1$.
- En algunos polinomios con los que otras versiones no tenían dificultades, *Mathematica* 5.1 concentraba —inexplicablemente— todos los ceros en el origen.

Y todo este caos, sin explicación alguna. ¿Cómo podemos fiarnos? Es más, ¿cómo puede, una mente tan racional como la de un matemático, no pensar que nos están tomando el pelo? Para no ser tan negativos, comentemos que, tras ardua búsqueda, descubrimos un truco que parece útil (y que posiblemente

pueda resultar provechoso en numerosas ocasiones). La idea es forzar a *Mathematica* a que, al resolver las ecuaciones, use aritmética de mayor precisión que la que emplea por defecto. Por ejemplo, si queremos que utilice números con 128 dígitos, recurrimos a la sintaxis `NSolve[z^n - 1 == 0, z, 128]`. Por lo que hemos podido experimentar, da la impresión de que no hacen falta muchos dígitos, pero sí que es útil evitar que use la precisión que tiene por defecto (aunque esto produce una pérdida de rendimiento, pues la trasmisión de los cálculos al microprocesador ya no es tan directa). Bueno ... , lo mejor es ser siempre cauteloso y, en consecuencia, recurrir a chequeos adicionales; los aquí descritos, o los que se le ocurran al lector ante el problema concreto que esté abordando.

11 Conclusión

Todo lo que hemos expuesto a lo largo de estas páginas han sido ejemplos académicos. No han ocasionado que un puente se derrumbe. A lo más, contratiempos al publicar un artículo de investigación, o al explicar algo en clase. ¿Pero qué hubiera pasado si uno de estos cálculos erróneos que hemos mostrado se hubiera cruzado con el cálculo de estructuras de una obra de ingeniería?

Las dificultades y consiguientes inexactitudes del cálculo numérico están asumidas desde hace tiempo. Cualquiera sabe que no se puede confiar a la ligera en un resultado numérico; hay que tomar precauciones. Pero el cálculo simbólico aún tiene poca tradición, y sus errores son inesperados. En muchas ocasiones, no hay ninguna razón que explique la aparición de resultados falsos. Son, simplemente, errores de programación; y, lamentablemente, existen. Si sabemos lo que estamos haciendo, es mucho más difícil que una respuesta errónea nos pase desapercibida. Los ordenadores no tienen malicia; si se equivocan, no saben disimular. Es habitual que las respuestas equivocadas carezcan de toda lógica para quien domina el tema.

Recomendamos al lector que sea precavido, pero no alarmista. Al fin y al cabo, todos podemos recordar el fallo informático más publicitado de los últimos tiempos: muchos programas sólo tenían en cuenta los dos últimos dígitos de cada año; al entrar el año 2000, éste se confundiría con el 1900, lo que podría ocasionar el mal funcionamiento de muchos sistemas informáticos. Se auguraban catástrofes pero, al final, no fue para tanto (aunque quizás debido a todo el dinero que se invirtió en prevenirlo).

Así pues, y pese a lo expuesto hasta ahora, queremos finalizar rompiendo una lanza a favor del uso de las potentes herramientas informáticas de las que ahora disponemos. No hay que explicar a nadie que los ordenadores han llegado a ser imprescindibles en todo tipo de trabajos técnicos y científicos. Sencillamente, muchas cosas no se podrían hacer sin su ayuda. Es más, ya nos hemos acostumbrado a usarlos en situaciones en las que no son estrictamente necesarios. No es dependencia; es que realmente son muy útiles.

Hasta tal punto es así que hace ya años que la informática ha venido contribuyendo a la matemática teórica. Queremos recordar que el famoso teorema de los cuatro colores –cada mapa se puede colorear, de modo que dos regiones adyacentes nunca estén pintadas igual, con sólo cuatro colores– fue demostrado, en 1977, con la ayuda imprescindible de un programa informático (véanse [3, 4]). Esto ocasionó una considerable controversia, pero no creemos que alguien piense ahora que tal prueba no es válida (aunque tampoco se puede considerar que sea una demostración sencilla y elegante, claro).

Sin llegar al extremo de pretender usar herramientas informáticas genéricas para demostrar teoremas –algo que quizás será más usual en el futuro–, lo que sí es habitual es que nos sirvamos de los ordenadores para efectuar cálculos de manera rápida y precisa. Incluso nos han proporcionado un laboratorio experimental, del que los matemáticos carecíamos. Los paquetes de cálculo simbólico están resultando ser muy útiles en todo esto; la ayuda que nos prestan es magnífica. Pero no son la panacea universal; hay que saber manejarlos y conocer sus limitaciones, a menudo inherentes a las matemáticas involucradas en el proceso. Además, este tipo de programas son aún muy nuevos, y no están adecuadamente depurados. Cometen errores achacables exclusivamente a sus programadores. Sólo los adecuados conocimientos matemáticos del usuario pueden ayudarle a detectar y mitigar estos problemas. De momento, los sistemas de álgebra computacional cuentan con un amplio margen para mejorar, y estamos convencidos de que así ocurrirá en el futuro; las numerosas y muy competentes personas que trabajan en este campo se ocuparán de ello. Ojalá que su precio y su opacidad disminuyan en la misma medida. Mientras, no queda otro remedio que desconfiar, ¡y rascarse el bolsillo!

Agradecimientos

Nuestra gratitud al profesor Julio Rubio, de la Universidad de La Rioja. Sus valiosos comentarios y sugerencias han logrado, sin duda, mejorar este artículo. No obstante, si algún *bug* queda, es exclusivamente achacable a los autores.

Referencias

[1] M. Abramowitz y I. A. Stegun (editores), *Handbook of mathematical functions with formulas, graphs, and mathematical tables*, United States Government Printing Office, 1964. Reimpreso por Dover, 1972 (10.ª reimpresión).

[2] M. P. Alfaro, M. Bello, J. M. Montaner y J. L. Varona, Some asymptotic properties for orthogonal polynomials with respect to varying measures, *J. Approx. Theory* **135** (2005), 22–34.

[3] K. Appel y W. Haken, The solution of the four-color-map problem, *Sci. Amer.* **237** (1977), 108–121.

[4] K. Appel y W. Haken, The four color proof suffices, *Math. Intelligencer* **8** (1986), n.º 1, 10–20.

[5] N. Blachman, *Mathematica: Un enfoque práctico*, Ariel informática, 1993.

[6] R. Bradford, R. M. Corless, J. H. Davenport, D. J. Jeffrey y S. M. Watt, Reasoning about the elementary functions of complex analysis, *Ann. Math. Artif. Intell.* **36** (2002), 303–318.

[7] R. Chapman, Evaluating $\zeta(2)$, 2003. Disponible en `http://www.maths.ex.ac.uk/~rjc/rjc.html`.

[8] J. H. Davenport, The difficulties of definite integration. Conferencia plenaria en el *11th Symposium on the integration of symbolic computation and mechanized reasoning "Calculemus 2003"* (Roma, 2003). Disponible en `http://www-calfor.lip6.fr/~rr/Calculemus03/`.

[9] C. W. H. Lam, How reliable is a computer-based proof?, *Math. Intelligencer* **12** (1990), n.º 1, 8–12.

[10] T. R. Nicely, Enumeration to 10^{14} of the twin primes and Brun's constant, *Virginia J. Sci.* **46** (1995), 195–204.

[11] I. Peterson, *Error fatal: A la caza de fallos informáticos*, Alianza Editorial, 1999.

[12] J. L. Varona, Graphic and numerical comparison between iterative methods, *Math. Intelligencer* **24** (2002), n.º 1, 37–46.

[13] J. L. Varona, Representación gráfica de fractales mediante un programa de cálculo simbólico, *Gac. R. Soc. Mat. Esp.* **6** (2003), 213–230.

[14] M. Wester, A critique of the mathematical abilities of CA systems. Capítulo 3 de *Computer algebra systems: A practical guide* (M. Wester, editor), John Wiley & Sons, 1999. También disponible en `http://math.unm.edu/~wester/cas_review.html`.

[15] S. Wolfram, *The Mathematica book*, 5.ª ed., Wolfram Media, 2003.

Óscar Ciaurri
Departamento de Matemáticas y Computación
Universidad de La Rioja
26004 Logroño, España
Correo electrónico: `oscar.ciaurri@dmc.unirioja.es`

Juan Luis Varona
Departamento de Matemáticas y Computación
Universidad de La Rioja
26004 Logroño, España
Correo electrónico: `jvarona@dmc.unirioja.es`
URL: `http://www.unirioja.es/dptos/dmc/jvarona/hola.html`