EPISTEMOLOGÍA E HISTORIA DE LA CIENCIA

SELECCIÓN DE TRABAJOS DE LAS XX JORNADAS

VOLUMEN 16 (2010)

Pío García
Alba Massolo

Editores

ÁREA LOGICO-EPISTEMOLÓGICA DE LA ESCUELA DE FILOSOFÍA
CENTRO DE INVESTIGACIONES DE LA FACULTAD DE FILOSOFÍA Y HUMANIDADES
UNIVERSIDAD NACIONAL DE CÓRDOBA

El diseño de simulaciones digitales: una perspectiva desde las prácticas científicas

Juan M. Durán[*] • *Penélope Lodeyro*[**] • *Maximiliano Bozzoli*[***]

1. Introducción

Durante los últimos cincuenta o sesenta años, la migración al uso de computadoras por parte de los científicos ha sido casi en su totalidad. Desde luego los usos y las aplicaciones son extremadamente variadas: desde procesadores de texto a analizadores de resultados, sólo por nombrar dos usos completamente opuestos. Pero ha sido sólo en estos últimos años cuando la filosofía ha comenzado a considerar más sistemáticamente el impacto que las computadoras pueden tener en la vida del laboratorio. Un plan de trabajo interesante para los filósofos consiste en evaluar el valor epistémico que se le adscribe a un programa de computadora en función de sus resultados. El problema inmediato con esta línea de trabajo consiste en suponer fuertemente que los resultados funcionan como evidencia para la validación del programa. Un caso particularmente interesante se presenta en el dominio de las simulaciones computacionales, especialmente en el área de física y disciplinas afines, donde algunos filósofos sostienen que la correspondencia entre los resultados de las simulaciones y los resultados de la experimentación son motivos suficientes para obtener garantías epistémicas sobre la simulación.

Nuestro trabajo consistirá en brindar otro enfoque a la propuesta basada en que los resultados son determinantes en la validación de una simulación, proponiendo un breve análisis filosófico acerca de la importancia epistémica que tienen algunos elementos técnicos en la validación del programa así como en los resultados. Desafortunadamente los límites conceptuales de lo que es una simulación computacional aun son extremadamente borrosos, por ello hemos optado por restringir este trabajo a lo que en matemática aplicada se denominan simulaciones numéricas (SN). En este sentido, entendemos por SN la resolución de un modelo matemático mediante métodos numéricos implementados en una computadora. Así pues, una SN se convierte en una subcategoría de lo que usualmente se llaman simulaciones computacionales, sólo que particularizada por el método de resolución de ecuaciones. El valor epistémico que estamos buscando estará, pues, relacionado íntimamente con la creación y uso de modelos computacionales, entendidos estos como una estructura lógica capaz de ser implementada en una arquitectura computacional; más específicamente, un modelo computacional será el producto de la aplicación de transformaciones

* UNC - Universität Stuttgart
**UNC - FONCyT
*** UNC - CONICET

formales a modelos matemáticos, más algún conjunto de conjeturas informales más o menos bien fundadas[1].

Dado este contexto, uno de los aspectos a tener en cuenta es la presencia inevitable de errores en los distintos niveles: en el modelo matemático, en la transformación y reducción de niveles de complejidad, en la especificación del modelo computacional, en el modelo computacional, en la implementación en una arquitectura específica, sólo por mencionar algunos. Así pues, el análisis del *error* resulta de gran interés filosófico para la validación de los resultados y, claro está, para la confianza epistémica depositada en la simulación. Aquí, por otra parte, sólo nos ocuparemos de trabajar el error en uno de los niveles mencionados, a saber, a nivel de modelo computacional.

2. Diseño de una simulación numérica: errores y validación interna

La noción de validación ha sido ampliamente tratada dentro de la filosofía de la ciencia y, en particular, en el contexto de las simulaciones computacionales. Por ello, y a los fines de este trabajo, nos limitaremos a una caracterización operativa basada en la distinción entre validación interna y validación externa. En el caso particular de las SN se dice que son *externamente válidas* cuando el modelo computacional representa, en un grado aceptable, el dominio estudiado. El interés que ha despertado en algunos filósofos la validez externa es extremadamente sugestivo, sobre todo en aquellas especulaciones que intentan disipar (o profundizar) el escepticismo que provocan las simulaciones computacionales como metodología científica para representar fenómenos del mundo (por ejemplo, Guala 1999, 2003; Morgan 2002, 2003, Morrison 2009).

Por otra parte, una SN es *internamente válida* si las soluciones del modelo numérico se aproximan, en el grado de precisión deseado, a las soluciones relevantes de las ecuaciones del modelo matemático original (Winsberg, 2003). El proceso de validación interna introduce legitimidad en el sistema, consistencia entre sistemas y no tiene implicaciones directas sobre confiabilidad del modelo para representar fenómenos. En este sentido, la validez interna de una SN puede fallar cuando los resultados reflejan artificios, ya sea por el modo en que fue formulado el modelo numérico, ya sea por el diseño o la implementación de la solución del algoritmo, o por alguna interferencia externa (como un golpe de tensión, por ejemplo). Es interesante notar que los modelos numéricos pueden tener componentes matemáticos cerrados que pueden verificarse[2]. Entendemos que una SN es *correcta* o está *verificada* si las soluciones[3] del modelo numérico, esto es, el algoritmo que resolverá la especificación, se aproximan asintóticamente al grado de precisión deseado. Claro que por esto último no nos estamos refiriendo a un agente competente que *decide* si los resultados de la simulación se aproximan a lo que se quería simular, sino más bien a que es técnicamente posible calcular el grado de error de una simulación[4].

Ahora bien, el mapa clásico de especificación e implementación de un algoritmo (Law & Kelton, 2000) supone la existencia de un fenómeno a modelar (estamos pensando principalmente

en problemas de la física, pero posiblemente el análisis podría extrapolarse a la economía, la biología o cualquier otra disciplina empírica) y de una especificación más o menos formal[5] de ese fenómeno (puede tomarse, por ejemplo, a la dinámica de fluidos como un problema real y a las ecuaciones Navier-Stokes como la especificación formal en un conjunto de ecuaciones parciales diferenciales). Esas ecuaciones son traducidas a un algoritmo[6] a través de la programación en un lenguaje, y luego ejecutadas en una computadora física como un programa. Este mapa presta la ayuda necesaria para fijar la atención tanto en el nivel de detalle del algoritmo, del programa y de su integración, como en el ámbito de su implementación y de su respectiva ejecución. En particular, nos concentraremos en el papel que juegan ciertos errores propios de la simulación numérica en estos dos niveles. Fijado este marco conceptual, ya podemos entrar en el análisis del error y su impacto epistemológico en los resultados de las SN.

Marie Farge, en (Farge, 2007), ha trabajado una clasificación de ciertos tipos de errores como epistémicamente relevantes. Su taxonomía alcanza varios puntos centrales en la caracterización del error en términos del riesgo que se corre de ser engañado por una representación falsa del fenómeno (lo que ella denomina *simulacrum*) (Farge, 2007, pp. 20). Sin embargo, parecería que esta taxonomía se limita sólo al nivel de la especificación del algoritmo en el mapa previamente descrito, cuando resulta también relevante incorporar la dimensión de la implementación del algoritmo y su ejecución en una computadora física. Ya que resulta innecesario repetir los argumentos de Farge, nos concentraremos en complementar su lectura con dos características propias de las SN que, entendemos, están directamente asociadas con un análisis del error: la compilación y la generación de números pseudo-aleatorios.

En su nivel más básico, una computadora sólo puede interpretar 0 y 1, de aquí que sea preciso algún mecanismo que medie como conector entre el algoritmo, entendido como un texto no-interpretado por la computadora, y la codificación de máquina, concebida como la implementación de dicho algoritmo en una computadora física.[7] Así, si un algoritmo funciona como intermediario entre las ecuaciones matemáticas originales discretizadas y la implementación en una arquitectura computacional particular, un compilador es un tipo de programa especial responsable por la "traducción" de este algoritmo al dominio del programa. El papel que juega en la vida de los programas es central y ha sido objeto de discusión de varios autores. En particular Rapaport ha señalado que existe una "laguna semántica" [semantic gap] entre el algoritmo y el programa que sólo puede ser llenada por el compilador: "¿Qué es esta laguna? Presuntamente las operaciones específicas como tipos de datos abstractos, etc. del lenguaje de alto nivel no se corresponden directamente con nada en el lenguaje de máquina. Pascal, por ejemplo, tiene el tipo de dato 'record', pero mi SUN probablemente no lo tiene. Así, el compilador es necesario para mostrar cómo construir o implementar 'records' en un lenguaje de máquina." (Rapaport, 2005)[8].

En el contexto de nuestro estudio, los errores de redondeo, de truncado[9] y afines no sólo deben ser tratados en un nivel teórico, como bien señala Farge, sino que también se les debe reconocer el correlato físico en la computadora, es decir, un error de este tipo debe ser tratado por el compilador para su implementación en la computadora física. Dado que un compilador es el encargado de "reconocer" la arquitectura física de la computadora, entonces también es el responsable del tratamiento de este tipo de error a un nivel más bajo. Desde luego que si a nivel algorítmico no se contuvo el problema de redondeo, difícilmente se le podrá exigir al compilador que rectifique el error, sin embargo el problema del redondeo no termina en la dificultad teórica mencionada por Farge sino que se traslada también a la implementación concreta del algoritmo. Si un compilador no es capaz de dar cuenta de errores de redondeo, poco importa que los sepamos manejar a nivel teórico. En particular, dado el lugar que ocupa, parecería que la importancia epistémica de un compilador trasciende el mero hecho de tratarse de un intermediario (como lo podría ser el sistema operativo, por ejemplo) instalándose, en la discusión, al mismo nivel que los problemas de valores iniciales, de cota y los ya mencionados errores de redondeo y truncado.

Un ejemplo ilustrará lo que queremos señalar. El experimento denominado *Lunar Laser Ranging* dejó una matriz retroflectora en la luna para que sea posible medir con cierto grado de exactitud la distancia de la tierra a la luna. Una vez que se obtuvieron los primeros resultados, dos grupos independientes analizaron los mismos datos. El comité de física gravitacional sabía que las 100 000 sentencias escritas en *Fortran* eran imposibles de verificar. Así que optaron por utilizar dos softwares desarrollados independientemente que permitiesen verificar el programa principal. Para sorpresa del comité, las cosas no salieron como las habían planeado:

> Una doble verificación de los errores en los modelos de software fueron provistos por un análisis de MIT-AFCRL de los datos sobre la distancia a la luna con software separados. Ambos análisis presentaron sus resultados satisfactoriamente. Sin embargo, en el momento en que MIT-AFCRL reportó sus resultados, nuestras soluciones erróneamente indicaban una amplitud de 1m para el término *Nordtvedt*. El error fue buscado en el truncado de algún término relativístico aparentemente insignificante en la ecuación de movimiento. Habíamos planeado sumar todos los términos relativísticos de orden $1/c^2$ antes de publicar nuestros resultados, pero no esperábamos que esas sumas afectasen los términos *Nordtvedt* significativamente (J G. Williams et al., p 553)

Esta anécdota ilustra el rol del truncado en un caso teórico donde el peso de la prueba recae sobre algún término que, en el momento, resultaba insignificante. La moraleja es que validar internamente una SN no sólo reduce la posibilidad de la existencia del error y del consiguiente espacio de soluciones, sino que reduce también la sistematicidad de un mismo error. Supóngase que los resultados fueran buenos, es decir, que el número de iteraciones no hubieran sido tal que introdujesen errores tan evidentes, quedaría pues la pregunta acerca de los fundamentos de la confianza depositada en el modelo. Este problema es semejante al problema filosófico de la

inducción: si validamos el modelo a partir de los resultados, sólo podemos garantizar que hasta *esa ejecución particular*[10] el modelo se comporta como se espera.

Como segundo punto a incorporar a la discusión, queremos analizar el papel que juega la generación de números pseudo-aleatorios en las SN. Como es sabido, no existen números puramente aleatorios en computación. Por un lado, si se conoce la "semilla" que inicia el motor de generación de números pseudo-aleatorios, entonces sería (teóricamente) posible conocer de antemano la secuencia total de números generados. Este hecho se demuestra fácilmente. el algoritmo es un proceso recursivo, computando una y otra vez el mismo cálculo, aunque con distintos valores de entrada. Si se conoce el primer número, y se conoce la función de cómputo, entonces se conoce toda la secuencia de salida de números pseudo-aleatorios.

La segunda razón por la cual usualmente no se considera que existan números puramente aleatorios es que dicha secuencia debe, por fuerza, repetirse después de generado cierto número de instancias. En efecto, así como los errores de truncado afectan a los resultados de las SN, también afectan el total de números pseudo-aleatorios generables. En este sentido, el límite de números pseudo-aleatorios posibles estará determinado por la memoria disponible.[11] A pesar de esta particular característica de los números pseudo-aleatorios, éstos no introducen, propiamente hablando, error en los resultados, sino engaño. La diferencia es muy sutil y podría explicarse con el siguiente ejemplo: si utilizo gran cantidad de variables pseudo-aleatorias para integrar una función, pero resulta que la distribución de esas variables es muy pobre, entonces se debería esperar que la integración sea igualmente pobre, aunque eso no implica que el proceso de integración es falso ni que, propiamente hablando, el resultado también lo sea.

Desde luego, pueden existir otras fuentes de error, en especial si se tiene en cuenta que una computadora consiste en una serie de "capas" de software y hardware que hace difícil un control total. Sin embargo, entendemos que los problemas de convergencia y estabilidad de los resultados de una SN están relacionados a estos pocos aspectos teóricos antes que a problemas espurios de hardware o de software.

¿Cómo afecta la presencia de errores en las SN? Esta pregunta tiene dos posibles respuestas, incidentalmente asociadas a la validez interna y externa. Dado que a nosotros sólo nos cabe responder qué sucede en el primer caso, sugerimos que si el error no se mantiene dentro de niveles aceptables, el poder epistémico de una SN se ve, cuanto menos, puesto en duda. Se dirá que si la validez interna falla, la representación del dominio no será buena, coexistiendo así los dos tipos de validez. Si bien no nos hemos pronunciado explícitamente sobre esto, sí hemos hecho uso de un ejemplo que parecería suponer cierto nivel de independencia entre una validez y otra[12]. En cualquier caso, el avance sobre herramientas matemáticas más sofisticadas así como en mecanismos más o menos híbridos, como generadores de números aleatorios basados en medición cuántica[13], parecería prestar apoyo a nuestra tesis.

3. Conclusión

El trabajo ha sido situado en un contexto en el cual se concibe a un modelo matemático como adaptado e implementado en una computadora digital mediante la aplicación de métodos numéricos sofisticados. En primera instancia, dentro del amplio dominio de las simulaciones computacionales, se han tomado bajo consideración las numéricas, únicamente. Esto se debe a que se ha intentado acotar el estudio del rol de ciertos errores numéricos a un prototipo de simulación, sin intención de reducir la problemática a sólo este tipo. Con este fin, se ha tomado en cuenta la justificación de la validez interna de las simulaciones numéricas, mencionando sólo la importancia de la validez externa a modo de contraste. El resultado fue una apuesta a complementar la taxonomía de Farge, incorporando la dimensión de la implementación del algoritmo y su ejecución en una computadora física en el tratamiento del error en la validación interna de las SN. En particular enfatizamos dos fuentes de errores internos: el compilador y los números pseudo-aleatorios.

En términos más generales y como propuesta abierta consideramos que la evaluación epistémica de una SN debe tener presente, entre otros factores, las particularidades que aquí hemos brevemente discutido. Como podemos apreciar a partir del análisis de los errores, la validez interna de una SN tiene como objetivo complementar el poder epistémico que se asume poseen los resultados de la simulación. Si se resta la posibilidad de validar internamente una SN, creemos que se estaría corriendo el riesgo de valorar (o sobrevalorar) epistémicamente los resultados y el modelo sin mayor fundamento que un correlato entre los resultados simulados y los obtenidos experimentalmente.

Notas

1 Para un análisis de la estructura conceptual involucrada en la creación de modelos computacionales a partir de modelos matemáticos véase (Winsberg, 1999, Humphreys, 2004)

2 La relación algoritmo-programa es muy estrecha: el programa P, cuando se corre en una máquina M, lleva a cabo el algoritmo A. La verificación de un programa consiste en brindar una prueba de que el P en M realizó A. Sin embargo, hay razones para dudar acerca del éxito que puede tener la verificación de programas como un método de aplicación general y completamente confiable para garantizar el funcionamiento de un programa, quizás la verificación de ciertos componentes cerrados del programa puede incrementar la confianza en la corrección de los teoremas más que brindar una demostración formal. Para una discusión detallada, véase DeMillo, R., Lipton, R., y Perlis, A. (1979) y Fetzer J. (1988)

3 Un aspecto importante de las SN es que, sin importar cuántas veces se ejecute la simulación, nunca se obtiene literalmente el mismo resultado. Pero dada la naturaleza probabilística de este tipo de simulaciones, es de esperar no contar con un marco determinístico sino, más bien, con una curva de error. Desde luego todas las soluciones aceptadas deben caer dentro de una curva de error mínimo aceptable.

4 Esto excluye el caso de la interpretacion de los resultados, que es una tarea netamente humana.

5 Este trabajo se encuentra lleno de "sub-problemas" que son líneas de investigación posibles. En este caso, cuando hablamos de especificación formal, estamos pensando en un sistema de ecuaciones, posiblemente ecuaciones sin resolución analítica posible.

6 No haremos mayores distinciones sobre lo que es un algoritmo. En este trabajo sólo nos interesa diferenciarlo de un programa. Se podrá considerar un algoritmo como una estructura lógica de funciones verificables mediante alguna semántica formal, y al programa como la implementación de ese algoritmo en un modo en que sea ejecutable en una máquina concreta (que puede ser física o abstracta) como la que menciona Fetzer en (Fetzer, 1998) O se podrá tomar al algoritmo como un procedimiento para computar una función y un programa como una implementación del algoritmo. Un proceso de computadora es un algoritmo siendo ejecutado. Es un dispositivo físico (una computadora) comportándose de un cierto modo, el modo es descrito por el programa, y el dispositivo físico ejecutando el proceso implementa el programa como en (Rapaport, 2005)

7 De ahora en más nos referiremos a la codificación de la máquina como programa.

8 En rigor Rapaport identifica cuatro relaciones que especifican una posible interpretación de esta "laguna semántica", ninguno de ellos introduce diferencia al punto que queremos señalar.

9 Un error de redondeo ocurre a nivel de precisión del número computado respecto de su valor exacto. Si la precisión no es muy buena entonces, por ejemplo, el cómputo puede divergir. Un error de truncado, por el otro lado, está vinculado al orden de discretización de un número. Todo número computable tiene una "longitud" máxima medida en la unidad de "palabra", si un número excede esa longitud será cortado en la i-ésima posición tal que sea computable por esa arquitectura.

10 Es importante recordar que en materia de computación, cambios de arquitectura, de distribución o inclusive de sistemas operativos o de compiladores pueden introducir todo un abanico de problemas.

11 Es interesante ver cómo el cambio de arquitectura y, por lo tanto, el cambio en el truncado de los números, también afecta la generación de números pseudo-aleatorios.

12 Nos referimos al ejemplo del *Lunar Laser Ranging*.

13 Véase (Pironio et al., 2010)

Bibliografía

DeMillo, R., Lipton, R. and Perlis, A. (1979) "Social processes and proofs of theorems and programs" *Communications of the ACM*. 22 (5). 271-280

Farge M. (2007) "Numerical Experimentation: A Third Way to Study Nature" en Kaneda Y, Kawamura H, Sasai M Frontiers of Computational Science. Springer. 15-30.

Fetzer J. (1988) "Program verification. The very idea" *Communications of the ACM*. 31(9).1063

Humphreys, P. (2004) *Extending Ourselves*, Oxford University Press.

Law, A. M. y Kelton, W. D. (2000) *Simulation Modeling and Analysis* McGraw-Hill.

Morrison, M. (2009) "Models, measurement and computer simulation. the changing face of experimentation" *Philosophical Studies, 143*, 33-47

Pironi, S. et al. (2010) "Random numbers certified by Bell's theorem" *Nature*, 464(7291).1021-1024

Rapaport, W. (2005) "Implementation is semantic interpretation: further thoughts" *Journal of Experimental & Theoretical Artificial Intelligence*. 17(4).385–417.

Williams, J. G. et al. (1976), "New test of the equivalence principle from lunar laser ranging", *Phys. Rev. Letter* 36

Winsberg, E. (1999) "Sanctioning Models. The Epistemology of Simulation", *Science in Context*, 12, 275-292.

Winsberg, E. (2003) "Simulated Experiments. Methodology for a Virtual World" *Philosophy of Science, 70*, 105-125