

SIMULACIÓN

Simulación implica crear un modelo que aproxima cierto aspecto de un sistema del mundo real y que puede ser usado para generar historias artificiales del sistema, de forma tal que nos permite predecir cierto aspecto del comportamiento del sistema.

En particular, usaremos computadores para imitar comportamientos del sistemas evaluando numericamente un modelo del mismo. Estas evaluaciones numericas son las que nos permiten *generar las historias artificiales* que no son mas que *experimentos*.

Modelo

Un modelo es una representación de un objeto, idea, o sistema en una forma diferente a la entidad misma. En nuestro caso el modelo es un conjunto de relaciones matemáticas o lógicas derivadas de supuestos sobre el comportamiento del sistema.

¿Para que?

Simulamos para explicar, entender o mejorar el sistema.

Ejemplo:

El diseño de un procesador involucra miles o millones de compuertas lógicas interconectadas. El proceso de crear el primer chip es sumamente costoso y no es posible darse el lujo de construir varios chips y luego verificar su funcionamiento. Lo que se hace es modelar el procesador y verificar su funcionamiento usando simulación.

¿Cuándo?

- a) El sistema real no existe. Es costoso, peligroso, consume mucho tiempo, o imposible de construir y experimentar con prototipos (nuevo computador o procesador, reactor nuclear).
- b) Experimentar con el sistema real es complicado, costoso, peligroso, o puede causar serios desajustes (sistema de transporte, sistema de manufactura, reactor nuclear).
- c) Necesidad de estudiar el pasado, presente, o futuro del sistema en tiempo real, tiempo expandido, o tiempo comprimido (sistemas de control a tiempo real, estudios en cámara lenta, crecimiento poblacional).
- d) Es sistema es tan complejo que su evaluación analítica es prohibitiva, bien sea porque el modelado matemático es imposible, o porque el modelado matemático no tiene solución analítica o numérica simple y practica (colas de espera, ecuaciones diferenciales no lineales, problemas estocásticos).
- e) Se puede validar satisfactoriamente el modelo de simulación.

Se podría decir “Simular cuando todo lo demás falla”, pero esto no es excusa para usar simulación inadecuadamente.

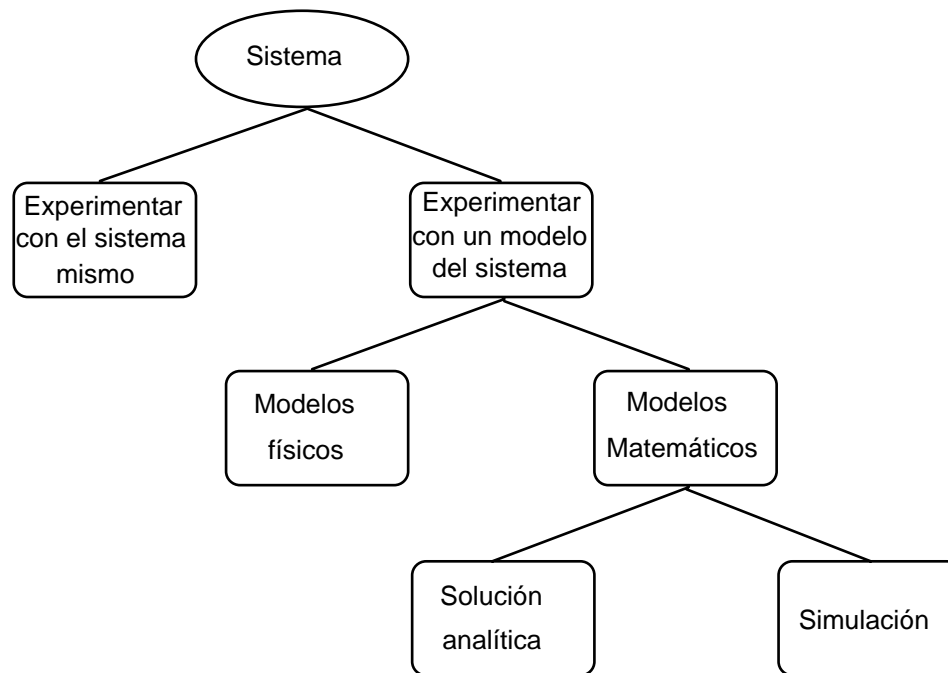


Figura 1. La simulación en el estudio de sistemas.

Las áreas de aplicación de la simulación son numerosas y entre ellas están:

- Diseño y análisis de sistemas de producción.
- Análisis de sistemas financieros o económicos.
- Evaluación de software y hardware.
- Evaluación de sistemas de armamento militar o sistemas tácticos.
- Determinación de políticas de inventario.
- Manejo de bosques.
- Diseño de sistemas de comunicación y protocolos.
- Diseño de sistemas de transporte.
- Evaluación de diseños de organizaciones como hospitales, comedores, servicios de correo, etc.

I. ERRORES COMUNES EN SIMULACIÓN

1. Nivel de detalle inapropiado

Un modelo analítico es menos detallado que un modelo de simulación. Análisis requiere de muchos supuestos y simplificaciones. El detalle en un modelo de simulación está limitado por el tiempo disponible para desarrollarlo.

Mas detalle ⇒ mas tiempo
 ⇒ incrementa la posibilidad de errores y es mas dificil detectarlos
 ⇒ incrementa el tiempo de corrida del modelo

¡Mas detalle no necesariamente es mejor!

Mas detalle requiere mas conocimiento de los parámetros de entrada, que si no están disponibles pueden hacer el modelo mas inexacto.

Ejemplo:

Supongamos que en la simulación de un sistema de tiempo compartido (timesharing) debemos simular el tiempo requerido para satisfacer accesos a disco. Una opción es generarlos usando una distribución exponencial. Una alternativa mas detallada seria simular el movimiento de los cabezales y la rotación del disco. En la segunda alternativa se pueden tener mejores resultados solo si conocemos las referencias a sectores y pistas. Sin embargo, si esta información no esta disponible a la hora de la entrada de datos, hay que terminar generándolos exponencialmente y hubiese sido menos costoso irse por la primera alternativa.

Es mejor partir de un modelo sencillo, obtener resultados, estudiar la sensibilidad, e introducir mas detalles en las áreas que impactan mas los resultados.

2. Lenguaje inapropiado

Lenguajes de simulación de propósito especial requieren menos tiempo para implementar el modelo y facilitan actividades como verificación (mediante el uso de opciones de trazado) y de análisis estadístico. Lenguajes de propósito general son mas portables y proveen mejor control sobre la eficiencia y el tiempo de corrida de la simulación.

3. Modelos no verificados

Los modelos de simulación son generalmente programas grandes, que si no se tienen las precauciones respectivas, es posible tener errores de programación que hagan las conclusiones sin sentido.

4. Modelos inválidos

Aun cuando no hayan errores de programación, puede que el modelo no represente al sistema real adecuadamente por supuestos incorrectos en su formulación. Es esencial que el modelo sea validado para asegurar que las conclusiones a las que se pueda llegar sean las mismas que se obtendrían del sistema real.

Todo modelo de simulación debe estar bajo sospecha hasta que se pruebe lo contrario por modelos analíticos, mediciones, o intuición.

5. Tratamiento incorrecto de las condiciones iniciales

Generalmente la parte inicial de una corrida de simulación no es representativa del comportamiento de un sistema en estado estable, por lo tanto debe ser descartada.

6. Simulaciones muy cortas

Por tratar de ahorrar tiempo de análisis y de computación, las corridas de simulación pueden ser muy cortas. Los resultados en estos casos dependen fuertemente de las condiciones iniciales y pueden no representar al sistema real. El tiempo de corrida adecuado depende de la exactitud deseada (intervalos de confianza) y de la varianza de las cantidades observadas.

7. Generadores de números aleatorios inadecuados

Las simulaciones requieren de cantidades aleatorias que son producidas por procedimientos llamados generadores de números aleatorios. Es mejor usar generadores que han sido bien analizados a usar los de uno mismo. Aun buenos generadores presentan problemas.

8. Selección de semillas inadecuadas

Los generadores de números aleatorios son procedimientos que dado un numero aleatorio generan otro. El primer numero aleatorio de la secuencia es llamado la semilla y debe ser proporcionada por el analista.

Las semillas para diferentes secuencias deben ser cuidadosamente seleccionadas para mantener independencia entre las secuencias. Los analistas usualmente usan una misma secuencia para diferentes procesos o usan la misma semilla para todas las secuencias. Esto introduce correlación entre los procesos y puede llevar a conclusiones erróneas.

II. OTRAS CAUSAS DEL FRACASO DE LOS ANÁLISIS DE SIMULACIÓN

1. Estimación inadecuada del tiempo para desarrollar el proyecto

Es común subestimar el tiempo y el esfuerzo requerido para desarrollar modelos de simulación. Si la simulación es exitosa y produce información útil, sus usuarios quieren incorporar mas funciones, parámetros y detalles. Por el contrario, si no provee de información útil, usualmente se espera que al añadir elementos la puedan hacer útil. En ambos casos el proyecto se extiende mas allá de las proyecciones iniciales.

Para proyectos grandes se deben hacer previsiones para incorporar cambios que son inevitables sobre largos periodos de tiempo.

2. Metas inalcanzables

La simulación es un proceso largo y complejo y se debe tener claramente definido un conjunto de metas que sean específicas, minuciosas, medibles, y alcanzables.

Un ejemplo común de una meta inalcanzable es "modelemos X." Es posible modelar muchas características diferentes de X a muchos niveles de detalle.

3. Mezcla incompleta de habilidades

Un proyecto de simulación requiere por lo menos:

- a. Liderazgo: Habilidad para motivar, guiar y manejar a los miembros del equipo de simulación.
- b. Modelaje y estadísticas: Habilidad para identificar las características claves del sistema y modelarlas al nivel de detalle requerido.
- c. Programación: Habilidad para escribir código entendible y verificable que implemente el modelo correctamente.
- d. Conocimiento del sistema modelado: Habilidad para entender el sistema, explicarlo al equipo de modelaje, e interpretar los resultados del modelo en términos de su impacto en el diseño del sistema.

4. Nivel inadecuado de participación de los usuarios

Es esencial que el equipo de simulación y los usuarios de la organización estén en constante contacto para intercambiar y discutir ideas. La mayoría de los sistemas evolucionan y cambian con el tiempo y un modelo desarrollado sin la participación de los usuarios raramente resulta exitoso.

5. Documentación inexistente u obsoleta

La mayoría de los modelos de simulación se desarrollan en largos periodos de tiempo y continuamente son modificados a medida que el sistema cambia o es mejor comprendido. Su documentación muchas veces es desatendida y rápidamente se vuelve obsoleta. Es recomendable documentar los programas y usar lenguajes que sean fáciles de leer.

6. Inhabilidad para gerenciar el desarrollo de programas de computación grandes

Hay muchas herramientas de ingeniería de la programación que permiten vigilar los objetivos del diseño, los requerimientos funcionales, las estructuras de datos y los estimados de progreso. También hay un conjunto de principios de diseño, como diseño de arriba abajo y programación estructurada, para desarrollar grandes proyectos en forma ordenada. Sin el uso de estas herramientas y técnicas es imposible desarrollar exitosamente un modelo de simulación grande.

7. Resultados misteriosos

Resultados misteriosos generalmente son debido a errores de programación, supuestos incorrectos en el modelo, o falta de entendimiento del sistema real. Nunca deben ser obviados.

III. TERMINOLOGÍA

1. Variables de Estado

El estado del sistema esta caracterizado por el valor que tengan las variables de estado en un instante de tiempo dado. El conjunto de variables de estado debe ser suficiente para el propósito de estudio, y puede diferir en el número y tipo de variables si los objetivos de la simulación cambian. Si la simulación es detenida, puede ser continuada después si y solo si los valores de las variables de estado son conocidos.

2. Evento

Un evento es un suceso, un hecho, una ocurrencia que genera un cambio en el estado del sistema. Una llegada, una salida, etc.

3. Modelos de Tiempo Continuo y de Tiempo Discreto

Modelo de Tiempo Continuo

Cuando el estado del sistema esta definido para cada instante de tiempo.

Modelo de Tiempo Discreto

Cuando el estado del sistema esta definido solo para particulares instantes de tiempo.

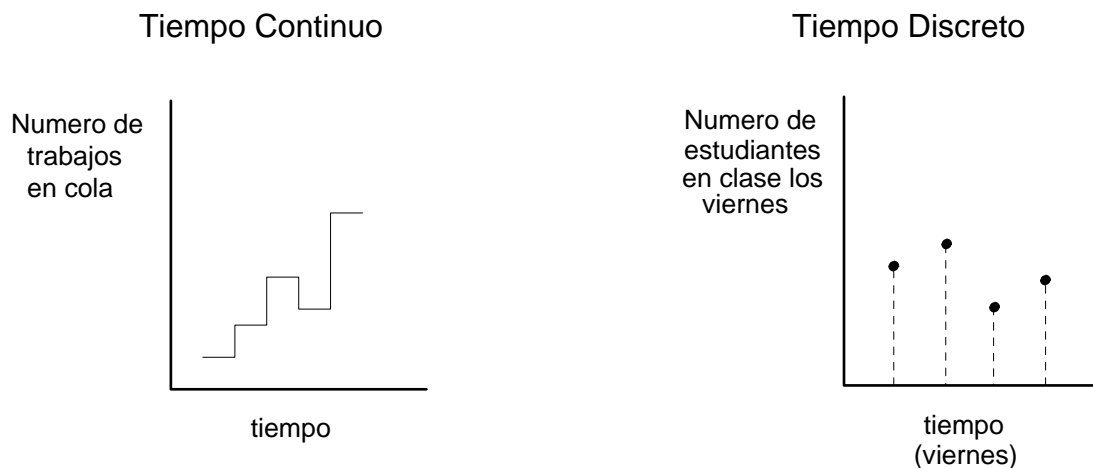


Figura 2. Modelos de tiempo continuo y discreto.

4. Modelos de Estado Continuo y de Estado Discreto

El modelo es de estado continuo o discreto dependiendo de si las variables de estado son continuas o discretas.

Modelo de Estado Discreto → Modelo de Eventos Discretos

Modelo de Estado Continuo → Modelo de Eventos Continuos

Continuidad de tiempo **no implica** continuidad de estado y viceversa.

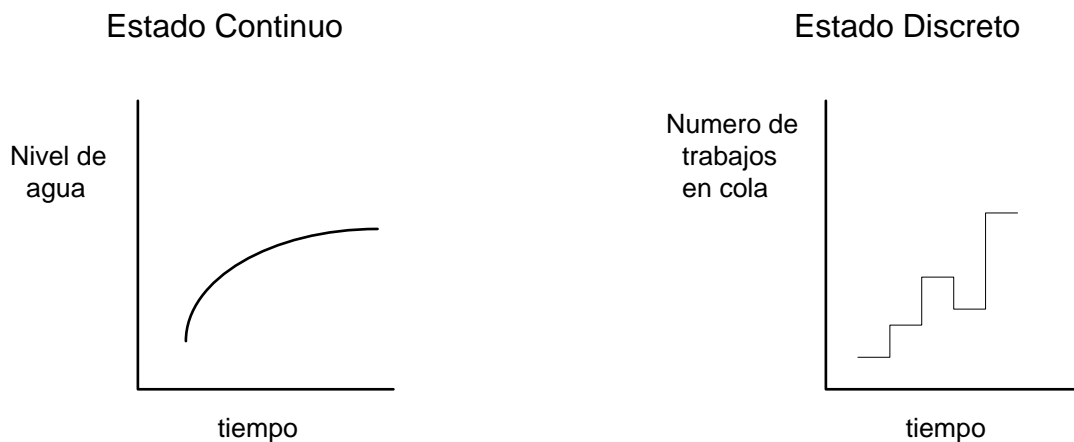


Figura 3. Modelos de estado continuo y discreto.

5. Modelos Determinísticos y Probabilísticos

Si los resultados de un modelo pueden predecirse con certeza, el modelo es determinístico; un simulador de contratos colectivos. Si repeticiones con la misma entrada pueden producir resultados distintos entonces el modelo es probabilístico; un simulador de colas.

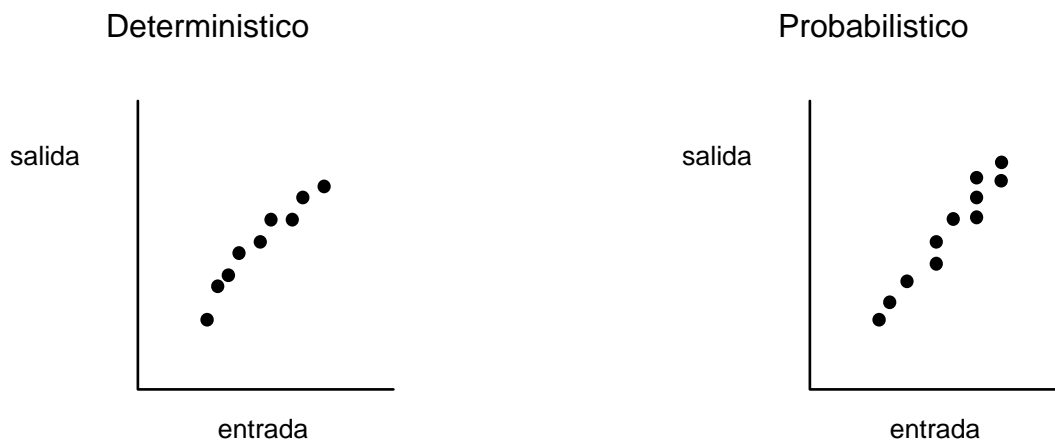


Figura 4. Modelos determinísticos y probabilísticos.

6. Modelos Estáticos y Dinámicos

Un modelo en el que el tiempo no es una variable es estático. Si el sistema cambia con el tiempo el modelo es dinámico.

$$E = m c^2 \quad \rightarrow \quad \text{Modelo Estático (materia en energía)}$$

Numero de trabajos en cola en un computador. \rightarrow Modelo Dinámico

7. Modelos Lineales y No-Lineales

Si la salida es una función lineal de la entrada, el modelo es lineal, de lo contrario es no-lineal. Ejemplo:

$$f(x) = a + bx \quad \text{vs.} \quad f(x) = a + b\sqrt{x}$$

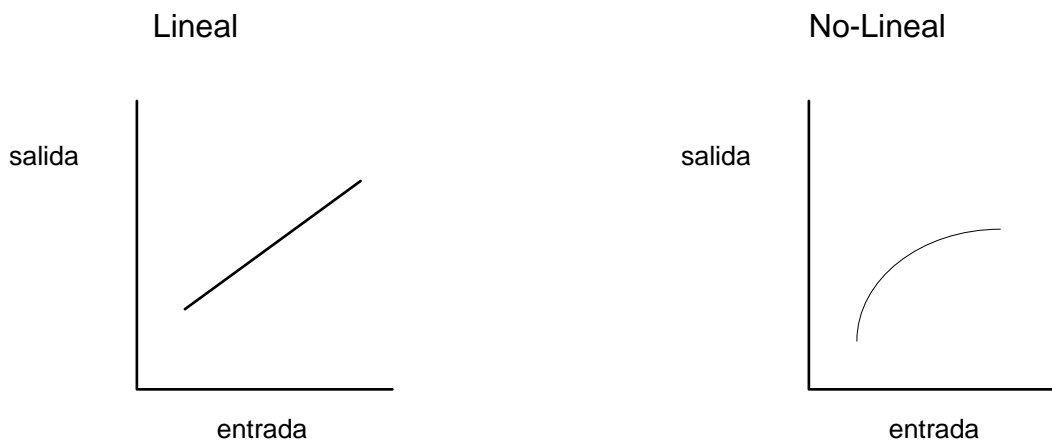


Figura 5. Modelos lineales y no lineales.

8. Modelos Cerrados y Abiertos

Si la entrada es externa al modelo e independiente de el, el modelo es abierto. Si el modelo es cerrado no hay entrada externa. Puede depender de los objetivos de la simulación, como se enfoque el problema y los supuestos que se hagan. Por ejemplo: podemos simular el trafico en una ciudad como un sistema cerrado si asumimos que el numero de vehículos permanece constante (no hay arribos ni salidas o arribos = salidas), o como un sistema abierto generando arribos y salidas al exterior.



Figura 6. Modelos abiertos y cerrados.

9. Modelo Estables e Inestables

Si el comportamiento del sistema converge a un estado estable (independientemente del tiempo) el modelo es estable. Un modelo cuyo comportamiento cambia constantemente es inestable. Por ejemplo: en un sistema de taquilla simple tenemos:

Intervalo entre llegadas $>$ tiempo de servicio \rightarrow modelo estable

Intervalo entre llegadas \leq tiempo de servicio \rightarrow modelo inestable.

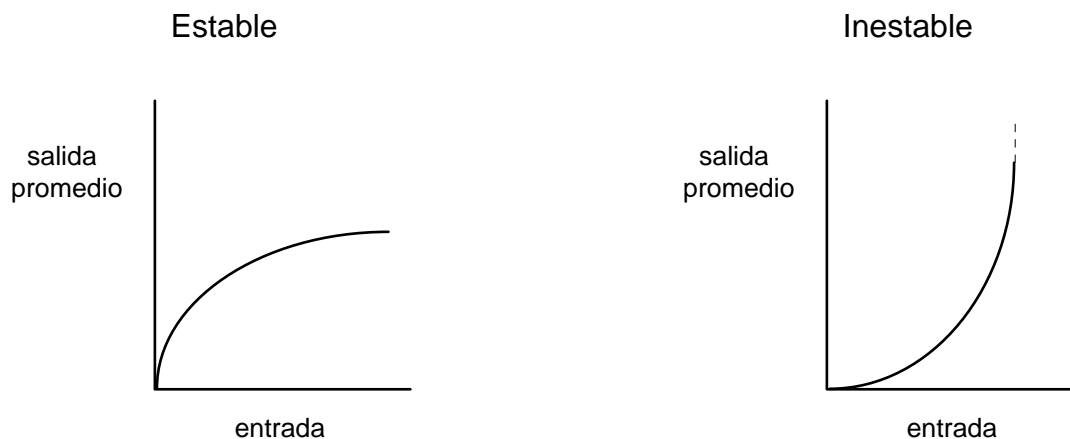


Figura 7. Modelos estables e inestables.

Ejemplo

Modelos de sistema de computación generalmente son de tiempo continuo, estado discreto, probabilísticos, dinámicos y no-lineales. Algunos son abiertos y otros cerrados. También pueden ser estables o inestables.

IV. SELECCIÓN DEL LENGUAJE DE SIMULACIÓN

1. Importante

La selección incorrecta del lenguaje de simulación puede extender considerablemente el tiempo de ejecución del proyecto, producir estudios incompletos o hacer fracasar el proyecto.

2. Lenguaje de Simulación

Lenguajes como SIMULA y SIMSCRIPT ahorran tiempo de desarrollo: tienen facilidades para generar estadísticas, reportes, etc. Permiten al analista concentrarse en aspectos específicos del sistema y no preocuparse por aspectos generales a todas las simulaciones. El código es modular, fácil de leer y proveen buena detección de errores.

3. Lenguaje de Propósito General

C, Pascal, Fortran. Se usan cuando el analista está familiarizado con el lenguaje, no hay tiempo de aprender un lenguaje de simulación o no está disponible. Proveen flexibilidad, eficiencia y portabilidad.

4. Extensiones a Lenguajes de Propósito General

Lenguajes de propósito general extendidos con un conjunto de rutinas comúnmente requeridas en simulación. Ejemplo: GASP para FORTRAN.

5. Paquetes de Simulación

QNET4 y RESQ permiten definir el modelo usando un diálogo. Pueden ahorrar mucho tiempo pero son muy inflexibles. Proveen solo cosas que fueron previstas por los desarrolladores.

| Continuos | Discretos | Mixtos |
|------------------|------------------|---------------|
| CSMP | SIMULA | SIMSCRIPT |
| DYNAMO | GPSS | GASP |
| | | GLIDER |

V. TIPOS DE SIMULACIÓN

1. Monte Carlo

Simulación estática o sin eje de tiempo. Se usa para modelar fenómenos probabilísticos que no dependen del tiempo o para evaluar expresiones no-probabilísticas con métodos probabilísticos. Esta definición es más restrictiva que la que dan otros autores para los cuales simulación Monte Carlo es cualquier simulación que use números aleatorios.

Ejemplo 1:

$$I = \int_0^2 e^{-x^2} dx$$

Para evaluar esta integral (para este ejemplo es mejor usar métodos numéricos y usar Monte Carlo en integrales múltiples con integrandos con comportamiento inusual) podemos generar números aleatorios con distribución uniforme x y para cada número calcular la siguiente función y :

$$x \sim \text{Unif}(0,2)$$

$$\text{función de densidad } f(x) = \frac{1}{2} \quad \text{si } 0 \leq x \leq 2$$

$$y = 2e^{-x^2}$$

El valor esperado de y es:

$$E(y) = \int_0^2 2e^{-x^2} f(x) dx = \int_0^2 2e^{-x^2} \frac{1}{2} dx = \int_0^2 e^{-x^2} dx = I$$

Y podemos evaluar la integral generando x_i , calculando y_i , y promediando de la siguiente forma:

$$x_i \sim \text{Unif}(0,2)$$

$$y_i = 2e^{-x_i^2}$$

$$I = E(y) = \frac{1}{n} \sum_{i=1}^n y_i$$

Programa y Corridas¹

| | | |
|--|-----------|------------------|
| <pre>program calcula_integral; var suma,n,x,y: extended; begin randomize; n := 0; suma := 0; while n < 1000000 do begin x := 2 * random; y := 2 * exp(-x * x); suma := suma + y; n := n + 1; end; writeln('Resultado: ',suma/n:6:4); end.</pre> | n | Resultado |
| | 10.000 | 0.8855 |
| | 100.000 | 0.8807 |
| | 1.000.000 | 0.8822 |

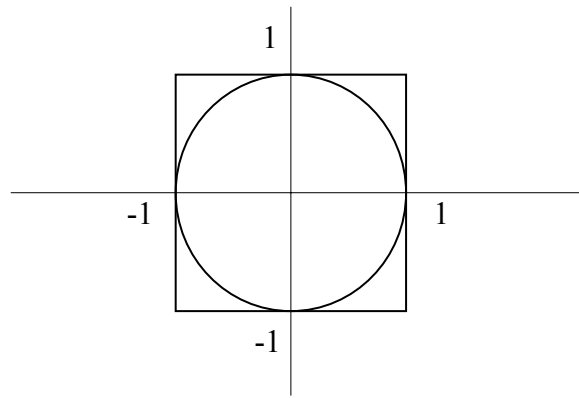
Notese que la funcion random genera números uniformes entre 0 y 1.

Ejemplo 2:

Para calcular π podemos basarnos en el hecho de que la relación entre el área del círculo y del cuadrado que se muestra a continuación es $\frac{\pi.r^2}{a.b} = \frac{\pi.1^2}{2.2} = \frac{\pi}{4}$. Por lo tanto podemos generar pares de

números aleatorios (x, y) entre -1 y 1 (en realidad es suficiente con generarlos entre 0 y 1 dada la simetría que presenta el problema y por lo tanto los resultados serán los mismos si consideramos solo el primer cuadrante), y calcular la relación entre los números dentro del círculo $d (x^2 + y^2 = 1)$ con respecto a los números generados n . Esta relación multiplicada por 4 debe aproximarse más π a medida que el número de pares generados sea mayor, es decir, $\lim_{n \rightarrow \infty} 4 \frac{d}{n} = \pi$

¹ Todos los códigos en Pascal fueron probados usando Delphi 5



Programa y Corridas

| | | |
|--|--|--|
| <pre> program calcula_pi; const limit = 10000000; var x, y, dentro, contador: extended; begin randomize; dentro := 0; contador := 0; while (contador < limit) do begin x := random; y := random; if (x*x + y*y) <= 1 then dentro := dentro + 1; contador := contador + 1; end; writeln('PI: ', 4.0*dentro/limit:10:8); end. </pre> | n 10.000 1.000.000 100.000.000 | Resultado 3.15560000 3.14124800 3.14139356 |
|--|--|--|

2. Simulación por trazas

Una traza es un registro de eventos ordenados por tiempo de un sistema real. Son muy usadas para analizar diferentes alternativas. Las trazas deben ser independientes del sistema bajo estudio.

Ejemplo

Para comparar diferentes esquemas de administración de memoria en un computador, se puede obtener del sistema una traza de paginas referenciadas por programas claves. Esta traza puede ser usada para ajustar los parámetros (tamaño de pagina) de un algoritmo de administración particular o para comparar diferentes algoritmos (FIFO, aleatorio, menos usada recientemente, etc.).

Ventajas

- a) *Credibilidad*: Una traza tiene mas credibilidad que una generación aleatoria.
- b) *Facilidad para validar*: Para obtener la traza hay que monitorear al sistema. Durante este monitoreo se pueden observar otras características del sistema que luego pueden usarse para validar la simulación.
- c) *Menos aleatoriedad*: Una traza es una entrada determinística. Si la simulación se repite, la traza es la misma pero la salida puede diferir por aleatoriedad en otros componentes del modelo. Hay

menos varianza en las salidas y por lo tanto se requieren menos repeticiones para obtener la confianza estadística deseada.

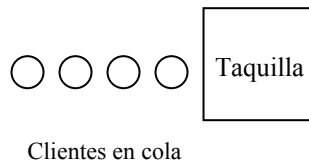
- d) *Comparaciones mas justas*: Una traza permite comparar diferentes alternativas con la misma entrada.

Desventajas

- a) *Complejidad y detalle*: Puede requerir una simulación mas detallada del sistema. Por ejemplo: en la simulación de un sistema de disco podemos asumir tiempo de acceso exponencial a sectores, o tomar trazas de los sectores accedidos e incluir cálculos del tiempo de rotación del disco y movimiento de cabezales ⇒ ¡mas cálculos y detalle!
- b) *Trazas finitas y representatividad*: En pocos minutos se puede obtener una traza detallada larga que puede no ser representativa en todos los momentos.
- c) *Validación*: Las trazas pueden producir diferentes resultados en diferentes alternativas. Deben usarse varias trazas distintas para validar los resultados.
- d) *Cambios de trazas*: Si se desea estudiar un sistema bajo condiciones distintas, no es posible modificar las trazas, y hay que tener trazas características para cada caso.

3. Simulación por Eventos Discretos

Una simulación que usa un modelo de estado discreto del sistema es llamada una simulación por eventos discretos. En este tipo de simulaciones el estado del sistema cambia solo en momentos específicos del tiempo: cuando ocurre un evento. Consideremos un sistema de taquilla simple:

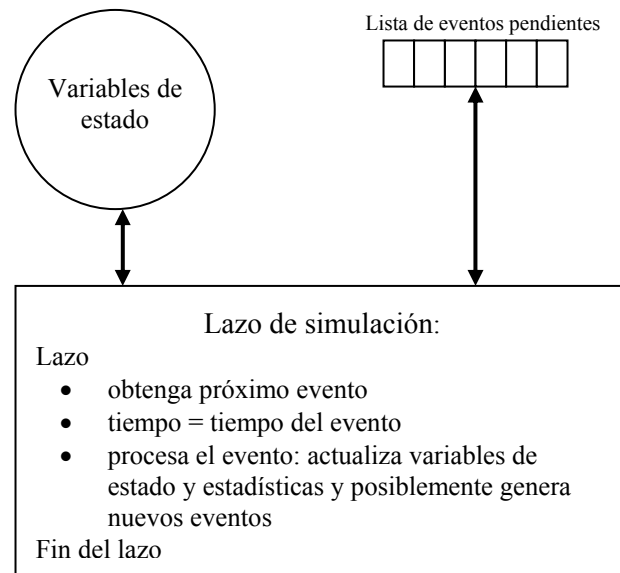


Aquí podemos considerar que la variable de estado (la única en este caso) es una variable entera, que podemos llamar cola, que indica cuantos clientes están frente a la taquilla con las siguientes consideraciones:

| Caso | Valor de cola |
|---|---------------|
| No hay clientes en el sistema | 0 |
| El primer cliente es atendido y los otros esperan | ≥ 1 |

En este tipo de sistema tenemos dos tipos de eventos, *llegadas* y *salidas*. Cuando un cliente arriba, la longitud de cola se incrementa (cambia el estado del sistema), y cuando un cliente termina de ser atendido este abandona el sistema y la longitud de cola se decrementa (vuelve a cambia el estado del sistema). Es claro que el estado del sistema no cambia constantemente y solo lo hace al ocurrir un evento: una *llegada* o una *salida*.

El esquema secuencial clásico de los simuladores por eventos discretos se muestra a continuación, en donde por lo general la simulación termina cuando el tiempo del próximo evento a simular es mayor que el tiempo que se estipulo para la simulación.



El simulador debe garantizar la *causalidad*, es decir, que todo evento sea simulado después de todos los eventos que él dependa. Para entender mejor esto, suponga una habitación con una puerta que esta cerrada y con una persona dentro. Si se tienen dos eventos: *abrir puerta* y *persona sale*, evidentemente no se puede simular la salida de la persona antes de simular la apertura de la puerta: el evento *persona sale* depende del evento *abrir puerta*. En un simulador en el que solo se permite que el tiempo avance (nunca retroceda) y en el que los eventos se simulen en estricto orden cronológico (los eventos más próximos en el tiempo primero), la causalidad esta garantizada. Si la causalidad no se garantiza se puede generar un caos y la simulación carecería de sentido alguno.

Todas las simulaciones de eventos discretos tienen una estructura común e incluyen los siguientes componentes:

- a) *Manejador de eventos:* Mantiene los eventos que esperan por suceder. Es de uno de los componentes de simulación más usados. Es ejecutado antes de la simulación de cada evento y posiblemente durante la simulación de un evento para programar nuevos eventos. Su implementación debe ser cuidadosa ya que tiene un fuerte impacto sobre la eficiencia del simulador.
- b) *Reloj de simulación y mecanismo de avance de tiempo:* Toda simulación tiene una variable global que representa el tiempo simulado. El *manejador* es el encargado de avanzar este tiempo. Hay dos formas de hacer esto:
 - Método de tiempo unitario:* Incrementa el tiempo en pequeños pasos y se chequea si hay eventos que pueden ocurrir. Generalmente no se usa.
 - Método por eventos:* Incrementa el tiempo automáticamente al tiempo de evento más próximo a ocurrir.

- c) *Variables de estado del sistema*: Son variables globales que describen el estado del sistema.
- d) *Rutinas de eventos*: Cada tipo de evento es simulado por su rutina. Estas rutinas actualizan las variables de estado y generan nuevos eventos.
- e) *Rutinas de entrada*: Son para obtener los parámetros del modelo como tiempo promedio entre llegadas, tiempo promedio de servicio, etc. Deben ejecutarse al comienzo de la simulación para liberar al usuario ya que generalmente las simulaciones consumen tiempo. Permiten variar los parámetros. Cada conjunto de valores de entrada define una iteración que quizás deba ser repetida varias veces con diferentes semillas. Cada ejecución de la simulación consiste de varias **iteraciones**, y cada iteración consiste de varias **repeticiones**.
- f) *Generador de reportes*: Rutinas para producir las salidas al final de la simulación.
- g) *Rutinas de inicialización*: Fijan el estado inicial del sistema e inicializan los generadores de secuencias de números aleatorios.
- h) *Rutinas de trazado*: Imprimen resultados intermedios durante la simulación. Sirven para depurar el simulador.
- i) *Manejo dinámico de memoria*: Durante la simulación se crean nuevas entidades y las viejas son destruidas. Esto requiere una periódica colección de basura. Si esto no lo provee el lenguaje, el programador debe escribir código para manejo dinámico de memoria.
- j) *Programa principal*: Agrupa todas las rutinas.

4. Simulación Continua

Los modelos de simulación continua, donde las variables de estado son continuas, usualmente son descritos mediante ecuaciones diferenciales y algebraicas. Estas variables de estado por lo general cambian en forma continua a medida que la simulación avanza, por ejemplo, el contenido de agua en un embalse cambia continuamente con la afluencia y salida de agua del mismo y no en solo en instantes específicos del tiempo como si estuviéramos agregando o sacando baldados de agua. Muchas simulaciones por eventos discretos incluyen subsistemas continuos y viceversa. Se puede hacer en computadores analógicos o en computadores digitales.

Computadores analógicos son maquinas de propósito especial:

- a) operan continuamente sobre todas las variables en tiempo real
- b) son dispositivos paralelos
- c) incluyen circuiteria para sumatorias, multiplicación, integración, y generación de funciones
- d) no tienen capacidades de almacenamiento o son muy limitadas
- e) dan resultados inmediatamente
- f) no pueden manejar variables muy grandes o muy pequeñas requiriendo cambio de escala
- g) las variables de entrada son continuas y esto requiere conversión de voltajes analógicos a números y viceversa.

Computadoras digitales son maquinas de propósito general:

- a) operaciones aritméticas y lógicas son efectuadas secuencialmente
- b) variables de entrada son discreteas
- c) pueden manejar números muy grandes o muy pequeños
- d) integración matemática directa no es posible y se hace usando métodos numéricos (ver el ejemplo siguiente)
- e) pueden almacenar gran cantidad de datos
- f) son relativamente lentas.

Ejemplo de integración numérica usando el método del trapecio

$$I = \int_0^2 e^{-x^2} dx$$

Programa y Corridas

El siguiente programa calcula el área bajo la curva usando trapecios como los mostrados en la figura 8.

| <pre> program calcula_integral; var suma,x,x1,x2,fx2,tria,paso: extended; begin paso := 0.1; x1 := 0; x2 := paso; suma := 0; while x2 <= 2 do begin fx2 := exp(-x2 * x2); tria := (exp(-x1 * x1) - fx2)*paso/2; suma := suma + tria + xf2 * paso; x1 := x2; x2 := x2 + paso; end; writeln('Resultado: ',suma:10:8); end. </pre> | <table border="1"> <thead> <tr> <th>paso</th> <th>Resultado</th> </tr> </thead> <tbody> <tr> <td>0.1</td> <td>0.87985</td> </tr> <tr> <td>0.01</td> <td>0.88208</td> </tr> </tbody> </table> | paso | Resultado | 0.1 | 0.87985 | 0.01 | 0.88208 |
|--|--|------|-----------|-----|---------|------|---------|
| paso | Resultado | | | | | | |
| 0.1 | 0.87985 | | | | | | |
| 0.01 | 0.88208 | | | | | | |

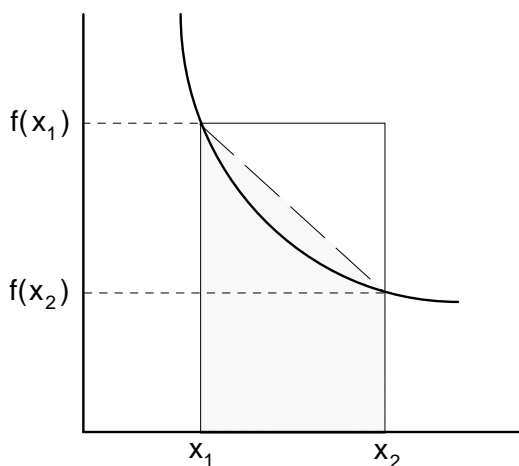


Figura 8. Integración numérica.

Ejemplo de un programa en CSMP y GLIDER

La distancia x , en metros, recorrida entre paradas por un ascensor en una mina esta dada por la siguiente ecuación diferencial:

$$x'' = -1.7(t - 2)^3 \quad \text{donde } t = \text{tiempo en segundos, } x'(0) = x(0) = 0 \quad \text{y } 0 \leq t \leq 4.$$

Calcular el valor de x como función del tiempo.

En **CSMP**, $Y = \int_0^t X dt + CI$; $Y(0) = CI$, tiene el siguiente formato $Y = \text{INTGRL}(IC, X)$.

| Programa | Comentarios |
|--|--|
| <pre>TITLE PROGRAMA EJEMPLO * SIMULA LA DISTANCIA RECORRIDA POR UN ASCENSOR CONSTANT K = -1.7 X2 = K*(TIME-2.0)**3 X1 = INTGRL(0.0,X2) X = INTGRL(0.0,X1) TIMER FINTIM = 4.0, PRDEL = 0.4, DELT = 0.01 PRINT X,X1,X2 END STOP ENDJOB</pre> | <pre>FINTIME tiempo de simulacion PRDEL intervalo entre impresiones DELT intervalo de integracion</pre> |

En **GLIDER** (Gate, Line, Input, Decision, Exit, Resource), se escriben directamente las ecuaciones diferenciales. Como no hay segunda derivada, se usa una variable auxiliar para especificarla.

| Programa |
|---|
| <pre>TITLE Distancia recorrida por un ascensor NETWORK C :: X' := Z; Z' := Y; Y := -1.7*(TIME - 2.0)*(TIME - 2.0)*(TIME - 2.0); IF (TRUNC(TIME*100) MOD 40 = 0) THEN ACT(IMP,0); ENC(A) :: WRITELN(F, ' TIEMPO X X1 X2'); WRITELN(F, '-----'); IMP(A) :: WRITELN(F, TIME:10:6, ' ', X:10:6, ' ', Z:10:6, ' ', Y:10:6); FIN(A) :: CLOSE(F); INIT ASSIGN(F, 'ELEV.OUT'); REWRITE(F); ACT(ENC,0); ACT(IMP,0); ACT(FIN,4.001); ACT(C,0); Z := 0.0; X := 0.0; Y := 13.6; TSIM := 4.001; DT_C := 0.01; DECL VAR X,Y,Z: CONT; F: TEXT; END.</pre> |

| Salida: ELEV.OUT | | | |
|-------------------------|-----------|----------|------------|
| TIEMPO | X | X1 | X2 |
| 0.000000 | 0.000000 | 0.000000 | 13.600000 |
| 0.400000 | 0.898373 | 4.047964 | 6.963200 |
| 0.800000 | 2.956270 | 5.972140 | 2.937600 |
| 1.200000 | 5.516312 | 6.689709 | 0.870400 |
| 1.600000 | 8.235768 | 6.856738 | 0.108800 |
| 2.000000 | 10.982108 | 6.868169 | -0.000000 |
| 2.400000 | 13.728557 | 6.857826 | -0.108800 |
| 2.800000 | 16.449645 | 6.698413 | -0.870400 |
| 3.200000 | 19.016759 | 6.001516 | -2.937600 |
| 3.600000 | 21.093697 | 4.117596 | -6.963200 |
| 4.000000 | 22.032218 | 0.136000 | -13.600000 |