IMPLEMENTACIÓN INFORMÁTICA DE LA TASA DE INTERÉS INTERBANCARIA MexIBOR

UNIVERSIDAD IBEROAMERICANA

Estudios con reconocimiento de validez por decreto presidencial del 3 de abril de 1981

IMPLEMENTACIÓN INFORMÁTICA DE LA TASA DE INTERÉS INTERBANCARIA MexIBOR

ESTUDIO DE CASO

Que para obtener el grado de

MAESTRO EN INGENIERÍA DE SISTEMAS EMPRESARIALES

Presenta:

LAURENCE RUIZ UGALDE

Director:
MTRO. ALFONSO MIGUEL REYES

Asesores:
MTRO. JORGE RIVERA ALBARRÁN
MTRO. EDUARDO BUSTOS FARÍAS

México, D. F. 2004

Índice general

Prefacio

Introducción

En el presente trabajo se muestra el diseño y operatividad de la tasa de referencia denominada MexIBOR, así como la implementación informática asociada a ésta, a modo de estudio de caso.

Contenido

En el capítulo 1 se ve de forma general las tasas de referencia de interés, y de forma particular la tasa MexIBOR.

Se muestran los organismos asociados a esta tasa, los detalles de su operación diaria y los elementos matemáticos asociados a su cálculo.

El capítulo 2 está dedicado al proceso de desarrollo del sistema informático MexIBOR. Se inicia con la discusión de dos de las tecnologías utilizadas en el sistema, que son la infraestructura de la compañía *REUTERS* y el sistema *Mathematica*. Esta discusión es muy importante puesto que buena parte de la implementación informática reside en la infraestructura proporcionada por estas tecnologías, de forma tal que interviene en la descripción técnica de los sistemas, desde sus etapas de arquitectura, diseño, hasta la implementación, operación y mantenimiento.

Se muestran con buen grado de detalle los elementos que conforman la arquitectura y diseño del sistema. Se enuncian brevemente algunos detalles propios de la implementación.

En el capítulo 3 se hace una discusión de la aritmética matemática y la computacional, su problemática y finalmente la aritmética racional.

Se hace un estudio no demasiado exhaustivo de los temas matemáticos, de estructuras de datos numéricas computacionales, teorías de números, de conjuntos y del álgebra necesarios para hacer uso de aritmética racional, que

fue necesaria para resolver problemas de exactitud en la operación informática del sistema MexIBOR, el cual es el tema del siguiente capítulo.

En el capítulo 4 se describe un error involucrado en el sistema informático, cómo fue detectado, analizado y posteriormente corregido.

El capítulo 5 es de conclusiones. Se presenta el sistema informático como un ejemplo de desarrollo de sistema de información en donde los principales retos fueron la integración de tecnologías, realización de sistema de tiempo real, de exactitud aritmética y de aspectos comúnmente difíciles en el proceso de diseño y escritura de programas. Se describen principios para utilizar aritmética racional en sistemas de información.

Todos los capítulo del presente trabajo contienen elementos de matemáticas, especialmente el primero y tercero. En el segundo capítulo predomina el lenguaje informático. Sin embargo se ha intentado que todo el contenido sea fácilmente asimilable para cualquier persona, sin importar su formación profesional.

Metodología de la Investigación

El presente trabajo es del tipo exploratorio, por ser la aritmética racional muy poco utilizada en sistemas computacionales de operación transaccional. Se pretenden encontar las condiciones bajo las cuales tanto las características del problema como el sistema mismo deben cumplir para verse beneficiados con este tipo de herramientas, y cómo puede aplicarse a otros sistemas.

Asimismo se pretende introducir al lector a los conceptos básicos de las tasas de interés de referencia, dado que su creación y difusión es poco frecuente.

Alcances y Limitaciones

No se presentan los programas fuente de los sistemas informáticos. Si bien estos programas fueron diseñados y escritos por el autor, los derechos de propiedad y uso pertenecen a *REUTERS Inc.*

Capítulo 1

Tasa de Interés Interbancaria MexIBOR

1.1. Tasas de Interés

Todas las operaciones crediticias requieren la aplicación de una tasa de interés. El valor de esta tasa no puede ser un valor arbitrario, si este estuviera determinado únicamente por la parte otorgante, tendería a ser alto, cayendo en situaciones de usurería; si este estuviera determinado por la parte solicitante, tendería a ser bajo, y para las instituciones de crédito dejaría de ser atractivo económicamente, los créditos no existirían y el ciclo financiero y crecimiento económico se verían fuertemente afectados. Tanto la parte otorgante como la solicitante deben estar de común acuerdo en el valor de esta tasa.

A fin de evitar estos desacuerdos entre transacciones crediticias, los paises crean tasas de interés de referencia, a ser utilizadas en transacciones formales, como las instituciones de crédito, cuya operación es regulada por organismos calificados para ello. Algunos de los países que determinan y utilizan tasas de interés de referencia son:

Australia	Eslovaquia	Japón
Alemania	España	Polonia
Argentina	Estados Unidos	Portugal
Brasil	Francia	Reino Unido
Canadá	Holanda	Rusia
Republica Checa	Hungría	Suiza
Chile	Italia	Zona Euro

Debido a que la tasa de interés en una operación crediticia además esta

regida por el plazo de la operación, estas tasas de referencia son generadas a diversos plazos.

Las tasas de interés de referencia reflejan las condiciones del mercado de dinero, y por este hecho, además de funcionar como referencia para operaciones de crédito, son utilizadas como base para otros tipos de instrumentos financieros.

Las dos tasas de interés típicas en México son la *Tasa de Interés Interbancaria de Equilibrio* (TIIE), y la *Tasa de Interés Interbancaria Promedio* (TIIP).

1.2. Tasa MexIBOR

El nombre se debe a que su mecanismo de operación y cálculos es similar a los de la tasa de referencia inglesa *London InterBank Offered Rate (LIBOR)*, en consecuencia la palabra MexIBOR es acrónimo de *MEXican InterBank Offered Rate*.

Su operación comenzó el día 2 de julio del año 2001. Se estima que la tasa MexIBOR irá sustituyendo gradualmente a las tasas TIIE y TIIP.

La tasa MexIBOR fue creada para obtener los siguientes beneficios:

1. Tener una tasa de interés de referencia de operación diaria para plazos en los cuales anteriormente no existían cotizaciones.

2. Estimular la liquidez en los mercados financieros al publicar una tasa de referencia que sea publicada de manera regular y que refleje las condicioens del mercado.

3. Promover la emisión de instrumentos financieros y el financiemiento de proyectos productivos de largo plazo.

4. Propiciar el desarrollo de nuevos productos financieros.

1.3. Organismos Involucrados en la Operación de la Tasa MexIBOR

1.3.1. Fideicomiso

La tasa MexIBOR funciona a través de un fideicomiso que tiene como propósito la creación de un vehículo legal responsable de la operación y la

administración del mecanismo de determinación de la tasa, así como por la aplicación de las políticas y reglas establecidas por el Comité Técnico. El fiduciario publicará diariamente los valores de las tasas en al menos cinco diarios de circulación nacional.

Las partes del fideicomiso son:

Fideicomitentes: Bancos participantes

Fiduciario: Nacional Financiera S.N.C.

Fideicomisarios: Bancos participantes.

La función de los bancos es cotizar valores de la tasa. Estos valores son sujetos a varios procesos matemático (en adelante *algoritmos*), con lo cual se obtienen los valores definitivos de la tasa.

1.3.2. Comité Técnico

A su vez, existe un *Comité Técnico*, integrado por un representante de cada uno de los bancos participantes y del Banco de México, el cual tiene voz, pero no voto. El Comité Técnico es responsable de formular las políticas, reglas y algoritmos para la operación y administración del fideicomiso, así como de supervisar el desempeño del fiduciario y el funcionamiento del sistema de cotizaciones.

1.4. Operación de la tasa MexIBOR

En esta parte se notarán los procedimeintos generales de operación y cálculo de la tasa MexIBOR. Los detalles de implementación se verán en el siguiente capítulo.

1.4.1. Periodo de Cotización

Cada día hábil bancario, a partir de cierta hora, los bancos participantes deben enviar al sistema central los valores de su cotización de la tasa para cada plazo. Actualmente los plazos para la tasa MexIBOR son doce, desde un mes, hasta doce meses.

Cada banco, durante el periodo de cotización no puede conocer los valores cotizados de cualquier otro banco.

1.4.2. Valores a Cotizar

¿Qué valores cotizar? por lo visto en la sección, a cualquier persona que le hicieran la pregunta respondería dependiendo de su rol en una operación de crédito. Para asegurar que la respuesta se encuentra determinada únicamente por las condiciones actuales de mercado, cada banco debería determinar su cotización en base a contestar la siguiente pregunta:

¿A que tasa de interés estoy dispuesto a *prestar* o a *pedir prestada* la cantidad x a otro banco (por plazo)? (la cantidad x de dinero esta previamente definida). Como es la misma tasa de interés la que sería utilizada en ambas posturas (otorgante y solicitante) se logra una situación de equilibrio.

1.4.3. Cálculo de la Tasa

Una vez que se tienen las cotizaciones de los bancos, estas son procesadas por los algoritmos para determinar el valor definitivo de la tasa, por plazo.

1.4.4. Penalizaciones

Teniendo los valores definitivos de la tasa, se pueden obtener diferencias absolutas entre los valores cotizados por un banco y los valores definitivos, para determinar el grado de desviación de cada banco respecto al resultado final.

Los algoritmos utilizan estas desviaciones para calcular penalizaciones monetarias para cada banco, por cada plazo. Estas penalizaciones están en función de la desviación presentada, a mayor desviación, es mayor la penalización. La penalización no puede ser mayor que cierta cantidad máxima.

Si un banco decidiera no cotizar, se le asignaría dicha penalización máxima por plazo.

Dentro de los varios parámetros de los algoritmos, existe uno denominado *diferencial*, que indica el rango dentro del cual la desviación presentada en un plazo respecto del valor definitivo es considerada como válida. Este rango se denomina *Rango de Tasa de Interés de Mercado* e indica hasta que umbral de desviación respecto del valor definitivo aún refleja las condiciones del mercado de dinero. Todas las cotizaciones de los bancos que se encuentren dentro de ese rango no serán penalizadas. Por ejemplo, si para el plazo de 3 meses el valor "diferencial" es de 3 puntos, y el valor definitivo de la tasa para dicho plazo resulta de 25 puntos, entonces todos los bancos que hayan cotizado en ese plazo valores entre 25 ± 3 puntos, o equivalentemente, en el rango $22 \ldots 28$, no recibirán penalización en ese plazo.

Evidentemente, La existencia de las penalizaciones tiene como fundamento evitar sesgo en los valores definitivos de la tasa provocados por valores cotizados poco uniformes, que pondría en riesgo la adherencia de la tasa a las condiciones de mercado.

Otro mecanismo de evitar cotizaciones sesgadas, incluídos como parámetros de los algoritmos, es eliminar la cotización mas alta y la mas baja por plazo, o las dos mas altas y las dos mas bajas. La decisión de no eliminar cotizaciones, eliminar una de cada extremo, o dos de cada extremo es reservada para el Comité Técnico (§1.3.2).

1.4.5. Periodo Extraordinario de Cotización (Segunda Ronda)

Si bien los algoritmos pueden operar con la cotización de un solo banco, el Comité Técnico se reserva la determinación de cuántos bancos como mínimo han de cotizar para declarar satisfactorio el proceso. Si en el periodo definido para cotizar no lo hicieron la cantidad mínima de bancos, se llega a un periodo extraordinario de cotización denominado "Segunda Ronda". Si esto sucede, todos los bancos son notificados de ello.

La duración de segunda ronda no necesariamente es la misma que la primera.

Todos los bancos que no hubiesen cotizado en el primer periodo (o "Primera Ronda") se les asigna una penalización fija extra.

Los bancos que hubiesen cotizado en la primera ronda, tienen la opción de no hacer nada en la segunda ronda (el valor que cotizaron en la primera ronda se mantiene durante la segunda), o cotizar nuevamente (posiblemente con valores diferentes).

1.4.6. Declaración de Operación Incompleta

Si se diese el caso de que existiese una segunda ronda, y aún después de esta, el número mínimo de bancos no hubiese cotizado, la operación del día se declara incompleta, y en tal caso, no existen valores para la tasa en dicho día.

1.4.7. Publicación

En caso de cálculo satisfactorio de la tasa, se procede a publicar[1] los resultados siguientes:

1. Los valores cotizados, los valores definitivos de la tasa, los valores diferenciales y las penalizaciones por plazo, a cada banco.

2. Los valores de la tasa, por plazo.

1.5. Algoritmos de Cálculo

Los algoritmos empleados para los cálculos de la tasa MexIBOR fueron aprobados por el comité técnico y sólo puede ser modificado por este comité. En caso de que algunos de estos algoritmos sean modificados, el fideicomiso deberá publicarlos inmediatamente en el diario oficial de la federación para la información de los intermediarios financieros y el público en general. La versión vigente de los algoritmos podrá ser consultada en la página de internet de la Asociación de Banqueros de México http://www.abm.com.mx

Dados que son varios los elementos a calcular alrededor de la tasa MexIBOR, se detallarán los algoritmos en base a lo siguiente:

1. Algoritmo de cálculo del diferencial.

2. Algoritmo de cálculo de las tasa MexIBOR.

3. Algoritmo de identificación de cotizaciones fuera de rango de mercado.

4. Algoritmo de cálculo de penalizaciones.

Cabe aclarar que el término *algoritmo* en este documento no tiene la connotación informática acostumbrada, en su lugar, considérese equivalente a *método matemático*.

[1] Esta "publicación" se refiere a un punto intermedio en todo el proceso cuyo final es el medio masivo de información, por ejemplo la prensa. En el apartado §2.2.2 del siguiente capítulo se detalla este proceso.

1.5.1. Algoritmo de Cálculo del Diferencial

Sea X_a el valor de la tasa MexIBOR en el día a. Entonces X_{a-1} es el valor de la tasa MexIBOR el día *hábil* anterior al día a. Por lo tanto, X_{a-i} es el valor de la tasa i días hábiles anteriores al día a.

Sea n el número de días que corresponden al periodo de observación.

Sea $\overline{X_a}$ es el promedio aritmético de las tasas registradas n días hábiles previos al día a:

$$\overline{X_a} = \frac{\sum_{i=1}^{n} X_{a-i}}{n} \tag{1.1}$$

Se obtiene la desviación estándar $\sigma(a)$ de las tasas MexIBOR, empleando como observaciones las tasas registradas durante los últimos n días hábiles previos a la fecha a en que se está haciendo el cálculo:

$$\sigma(a) = \sqrt{\frac{\sum_{i=1}^{n} \left(X_{a-i} - \overline{X_a}\right)^2}{n}} \tag{1.2}$$

Se obtiene el diferencial convencional D_c como el promedio móvil de las desviaciones estándar de las tasas MexIBOR correspondientes a los n días hábiles previos a la fecha en que se está haciendo el cálculo.

$$D_c = \frac{\sum_{i=1}^{n} \sigma(a-i+1)}{n} \tag{1.3}$$

El valor del diferencial nunca debe ser menor de un diferencial mínimo D_{min}, por lo que el valor definitivo del diferencial D es:

$$D = max\{D_c,\, D_{min}\} \tag{1.4}$$

1.5.2. Algoritmo de Cálculo de la Tasa MexIBOR

Sea k el número de cotizaciones presentadas por los bancos. Se obtiene el vector $\overrightarrow{Xa}$ que contiene las k cotizaciones ordenadas de menor a mayor:

$$\begin{aligned} \overrightarrow{Xa} &= (X_1,\, X_2,\, \ldots,\, X_k) \\ X_1 &\leq X_2 \leq \ldots \leq X_k \end{aligned} \tag{1.5}$$

Nótese que a diferencia del algoritmo anterior, en dónde X_i es el valor de la tasa para el día i, en este algoritmo el valor X_i significa el valor de i-sima

cotización, de entre k cotizaciones del mismo día, cada una debida a diferente banco.

Asimismo, el vector $\overrightarrow{Xp}$ contiene las mismas cotizaciones pero ordenadas de mayor a menor:

$$\overrightarrow{Xp} = (X_k,\, X_{k-1},\, \ldots,\, X_1) \tag{1.6}$$

El diferencial D (§1.5.2) es sumado a todos los componentes del vector $\overrightarrow{Xa}$, y restado a todos los componentes del vector $\overrightarrow{Xp}$, obteniéndose los vectores $\overrightarrow{Xa'}$ y $\overrightarrow{Xp'}$, respectivamente:

$$\begin{aligned} \overrightarrow{Xa'} &= (X_1 + D,\, X_2 + D,\, \ldots,\, X_k + D) \\ &= (Xa'_1,\, Xa'_2,\, \ldots,\, Xa'_k) \end{aligned} \tag{1.7}$$

$$\begin{aligned} \overrightarrow{Xp'} &= (X_k - D,\, X_{k-1} - D,\, \ldots,\, X_1 - D) \\ &= (Xp'_1,\, Xp'_2,\, \ldots,\, Xp'_k) \end{aligned} \tag{1.8}$$

Sea u el número de componentes positivos del vector de diferencia:

$$\overrightarrow{Xp'} - \overrightarrow{Xa'} = (Xp'_1 - Xa'_1,\, Xp'_2 - Xa'_2,\, \ldots,\, Xp'_k - Xa'_k) \tag{1.9}$$

Definimos las tasas r_1 y r_2 como:

$$r_1 = \begin{cases} max\{Xa'_u,\, Xp'_{u+1}\} & 0 < u < k \\ Xa'_1 & u = 0 \end{cases} \tag{1.10}$$

$$r_2 = \begin{cases} min\{Xa'_{u+1},\, Xp'_u\} & 0 < u < k \\ Xp'_1 & u = 0 \end{cases} \tag{1.11}$$

El valor de la tasa MexIBOR T es el promedio aritmético de las tasas r_1 y r_2:

$$T = \frac{r_1 + r_2}{2} \tag{1.12}$$

1.5.3. Algoritmo de Identificación de Cotizaciones Fuera de Rango de Mercado

Una cotización de un banco participante se considerará fuera de rango de mercado si cumple cualquiera de las condiciones siguientes:

$$cotizacion < T - D \tag{1.13}$$
$$cotizacion > T + D \tag{1.14}$$

O equivalentemente, que la cotización no se encuentre dentro del rango $T \pm D$, o fuera del intervalo $[T - D, T + D]$.

1.5.4. Algoritmo de Cálculo de Penalizaciones

La *Penalización convencional* P_c, es calculada como:

$$P_c = MB \times min\{|T + D - C|, |T - D - C|\} \times FP \times \frac{P}{360} \tag{1.15}$$

en donde:

MB Es el monto base.

D Es el diferencial calculado (§1.5.1).

T Es el valor de las tasa MexIBOR calculada (§1.5.2).

C Es la cotización provista por el banco.

FP Es el factor de pago.

P Plazo, expresado en días.

Sin embargo, esta penalización nunca podrá ser mayor que una valor máximo P_{max}, ni menor a un valor mínimo P_{min}, por lo tanto:

$$penalizacion = \begin{cases} P_{max} & P_c > P_{max} \\ P_c & P_{min} \leq P_c \leq P_{max} \\ P_{min} & P_c < P_{min} \end{cases} \tag{1.16}$$

El comité técnico determinará y podrá modificar en cualquier momento los valores de los días del periodo de observación, el diferencial mínimo, el monto base, el factor de pago, la penalización máxima y mínima.

De acuerdo a lo anterior, el comportamiento de las penalizaciones es como se muestra en la figura 1.1.

Se puede apreciar el hecho de que la penalización es 0 cuando la diferencia entre el valor cotizado y el de la tasa es menor o igual que el diferencial (dentro

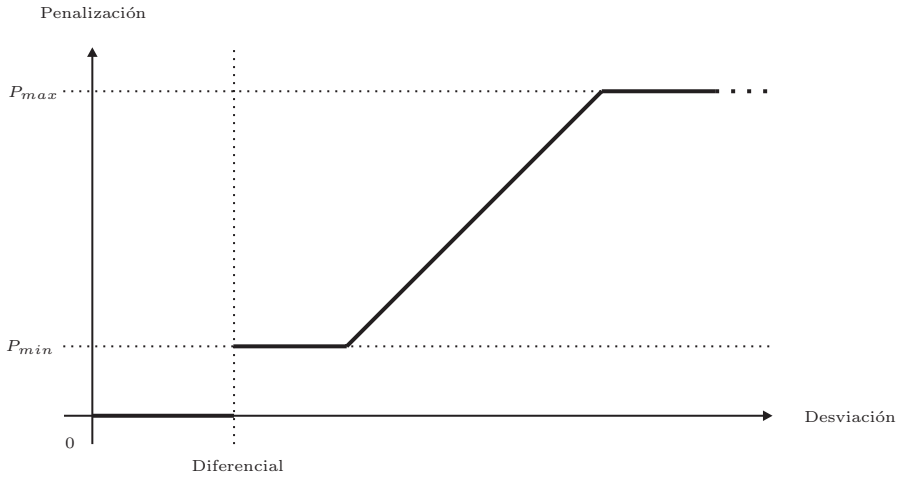

Figura 1.1: Penalizaciones en el Sistema MexIBOR, respecto a la desviación entre lo cotizado y valor de la tasa.

del rango del mercado). En caso contrario su comportamiento es directamente proporcional (lineal), a excepción de que se encuentra acotado por valores mínimos y máximos de dicha penalización.

Capítulo 2

Implementación Informática

2.1. Tecnologías Integradas en el Sistema

2.1.1. Infraestructura *REUTERS*

La compañía *REUTERS Inc.* es una empresa internacional dedicada a vender y comprar información. De todos los tipos de información que comercializa, es más conocida por los siguientes:

- Noticias
- Información Financiera de todo tipo (bursátil, divisas, futuros, mercancias, derivados, etc.).

Estos dos grandes grupos comparten una característica común muy especial: su valor económico (comercial) decrece exponencialmente en el tiempo, esto es: su valor mas alto es durante los primeros segundos en que se sucede, y al cabo de un tiempo corto, típicamente minutos, su valor prácticamente se nulifica.[1]

Es por lo tanto de suponerse que *REUTERS*, tenga como parte de su estrategia comercial primaria, el contar con una infraestructura de obtención y entrega de información eficiente. Además de eficiente esta entrega de información debe ser confiable, puesto que muchas empresas del sector financiero y de noticias basan parte o toda su operación en dicha información.

[1] ¿Alguna vez el lector ha observado que algunas páginas *web* ofrecen valores numéricos o gráficas de información financiera de acciones, divisas, etc.? Al observar mas detenidamente se dará cuenta que se esta se presenta con retraso de algunas decenas de minutos.

Se decidió entonces que el mecanismo de transporte de datos hiciera uso de la infraestructura *REUTERS* y asi aprovechar su eficiencia y confiabilidad. Asimismo, la operación informática central se lleva a cabo en las instalaciones de *REUTERS de México*, a su vez, por neutralidad en el proceso.

Arquitectura Informática Basada en Páginas

El mecanismo típico de intercambio de información que comercializa *REUTERS* lo hace a través de una abstracción denominada *página*. Su nombre se debe a que parece una hoja de papel cuadriculado, que puede escribirse en ella y leerse.

Una página cuenta con permisos de lectura y escritura. Sólamente pueden cambiar o actualizar su contenido aquellas entidades con permiso de escritura. Una página sólamente puede ser vista por aquellas entidades que tengan permiso de lectura de la misma.

La operación típica es conceder permiso de escritura a ciertas páginas a entidades a quienes compra información, y conceder permiso de lectura a ciertas páginas a entidades a quienes vende información, si bien no es el único esquema de trabajo utilizado. En este trabajo se presentará una arquitectura diferente.

Existe una infraestructura distribuída para administrar usuarios, páginas, y permisos entre ellos.

El acceso a las páginas se hace a través de aplicaciones desarrolladas por *REUTERS* para visualizar páginas (con permiso de lectura), o editar páginas (con permiso de escritura). Esta es la funcionalidad básica, pero existen programas especializados para tareas más específicas, por ejemplo uno de ellos dedicados a auxiliar a los intermediarios de acciones bursátiles, podría presentar gráficas de comportamiento, o emitir alarmas sonoras ante sucesos especiales, por ejemplo, que el valor de cierta acción se coloque por debajo o arriba de ciertos valores predefinidos.

Existe sin embargo, la posibilidad de escribir programas que tengan la capacidad de leer/escribir en páginas (con permisos para ello) a través de *APIs (Aplication Programming Interfase)*, que son librerías de desarrollo creadas por *REUTERS*.

Existen varias de estas *APIs*, algunas para proveer este servicio en ciertos entornos operativos, otras destinadas a proveer facilidad de uso al programador.

2.1.2. El Sistema *Mathematica*

Mathematica es un sistema de matemática simbólica, es decir, además de realizar operaciones aritméticas, puede realizar operaciones que requieren manipulación simbólica, como en álgebra o cálculo infinitesimal.

La utilización típica del sistema *Matematica* es en modo *de comando*, en la cual se tiene una pantalla[2] en donde el usuario escribe lo que desea que haga el sistema, por ejemplo una operación aritmética, una solución de ecuación, un gráfico de una función, etc., el sistema *Mathematica* lo realiza o resuelve, presenta el resultado al usuario y queda listo para otra petición.

Existen *APIs* para integrar *Mathematica* con otros lenguajes de programación. La interacción entre ambos es muy similar al modo de comando, en donde el lugar de un usuario humano lo toma el programa computacional. Dicho programa abre una sesión con *Mathematica* y a través de ella, realiza sus peticiones y obtiene los resultados en forma síncrona[3].

Se denomina *kernel* al subsistema *Mathematica* que es el que realmente realiza todos los cálculos. El entorno de comandos de *Mathematica* es el medio de interacción del usuario con dicho *kernel*. Los *APIs* antes mostrados son también un medio de comunicación con el *kernel*, pero en lugar de un usuario humano, es un programa de computación.

El *API* utilizado para el desarrollo del sistema MexIBOR es llamado *MathLink*, para integración con el lenguaje C/C++.

2.2. Arquitectura del Sistema MexIBOR

2.2.1. Subsistemas Principales

El sistema se encuentra dividido en dos grandes subsistemas.

El Sistema Central Es la aplicación que, diariamente recibe las cotizaciones de los bancos, ejecuta los algoritmos con estas para obtener los valores de las tasas y las penalizaciones y publica los resultados.

El Sistema de los Bancos Es la aplicación instalada en los bancos, cuya función es proveer el medio por el cual el representante del banco hace llegar sus cotizaciones hacia el sistema central.

[2] O una ventana en entornos gráficos.

[3] Esto significa que el programa tiene que esperar a que *Mathematica* obtenga el resultado para obtenerlo. Asimismo, no puede realizar una nueva operación si no se han obtenido los resultados de la anterior. También puede funcionar de forma *asíncrona*.

2.2.2. Transporte de Datos

Las cotizaciones de los bancos deben llegar hacia el sistema central para poder calcularse la tasa. Aprovechando el mecanismo de manipulación provisto por *REUTERS* a través de páginas, se definen varias de ellas:

Páginas de los Bancos Existe una de ellas por cada banco participante. El Banco tiene permisos de lectura y escritura a su página, pero no cuenta con permisos para acceder la demás. El sistema central tiene permisos de lectura y escitura a todas las páginas de los bancos.

Páginas MexIBOR En estas páginas se publican los resultados definitvos de la tasa, una vez que se ha calculado, así como las penalizaciones. Sólamente el sistema central cuenta con permisos de escritura y lectura a esta página. Las entidades que tienen acceso a esta página (de lectura sólamente) son aquellas que tomarán la información para su publicación masiva, por ejemplo, los editores de diarios. Su único fin es la publicación de resultados; Existen varias de ellas debido a que la cantidad de información no cabe en el tamaño de una sóla página.

Página de Alerta Como puede verse, no existe un mecanismo de comunicación directa entre el sistema central[4] y los sistemas de los bancos. Sin embargo, la coordinación entre ellos es vital. Esta página existe para efectos de coordinar tiempos entre ellos, por ejemplo para notificar que el periodo de cotización comienza. Sólamente el sistema central tiene permisos de escritura en esta página. Sólamente los bancos tienen acceso a esta página (de lectura).

Dado lo anterior, arquitectónicamente el sistema MexIBOR es como en la figura 2.1.

2.3. Diseño del Sistema MexIBOR

2.3.1. Funcionalidad Orientada a Estados y Eventos

El sistema MexIBOR, por diseño, trabaja bajo abstracciones denominadas *estados* y *eventos*:

[4]Como podría suceder por ejemplo, utilizando un protocolo de red punto a punto, como TCP/IP, por ejemplo).

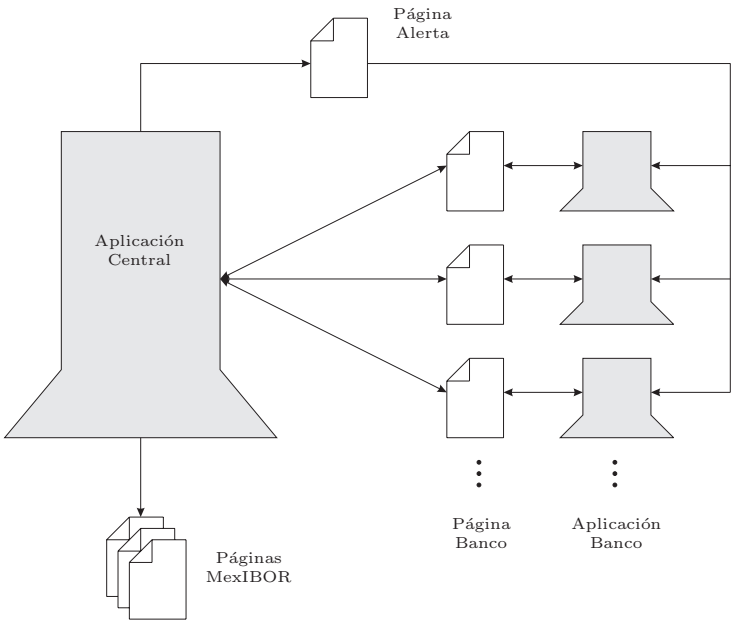

Figura 2.1: Diseño Arquitectónico del Sistema MexIBOR

Un Estado es una división o un periodo de tiempo. Existen varios estados posibles, y el sistema sólamente puede encontrarse en uno y sólo un estado en un instante cualquiera del tiempo.

Un Evento es un suceso importante, al cual el sistema responde de una manera bien definida. A diferencia de un estado, que goza de una duración, un evento debe considerarse como de duración infinitesimal, no importante o despreciable.

Cada día el sistema crea un archivo de texto, denominado de *bitácora* en el que se registran todos los eventos y cambios de estado que se suceden en el sistema, así como los resultados del cálculo de la tasa.

Estados

Debido a la naturaleza del equipo humano, organismos y tecnologías utilizadas para el desarrollo del sistema MexIBOR, todos los conceptos y la interfaz del sistema están en idioma inglés, a excepción de los manuales. Debido a ésto, muchos de los conceptos y nombres utilizados en el capítulo 2 se han dejado

en este idioma, aún cuando en la mayoría de los casos exista una traducción literal apropiada, aún en los manuales.

A continuación se enumeran los diferentes estados de la aplicación:

Standby Es el estado inicial del sistema. En este estado se hacen todos los preparativos pertinentes para el funcionamiento correcto del programa. Este estado se inicia invariablemente a las 0:00 horas de cada día.

Round1 En este estado los diferentes bancos pueden cotizar, y las cotizaciones que emitan son utilizadas para el cálculo de la tasa y de las penalizaciones tal y como se van recibiendo. La hora de inicio y duración de este estado es definida por el usuario.

Round2 Este estado es alcanzado únicamente cuando, al término del periodo de cotización *Round1* no se tengan cotizaciones de un número mínimo de bancos (este número es definido por el usuario). La duración de este estado es configurable por el usuario.

Final Es el estado de cierre normal (satisfactorio). Sólo se puede llegar a este estado si el número mínimo de bancos han cotizado al final de *Round1* o *Round2*. Se calculan las penalizaciones y tasas. Estos valores, junto con los valores de las tasas ingresadas por los bancos se escriben en las páginas MexIBOR (publicación de la información). Este estado termina invariablemente a las 24:00 horas.

Incomplete Es el estado de cierre no satisfactorio. Este estado es alcanzado únicamente si al final del estado *Round2* el número mínimo de bancos aún no ha cotizado. En este estado se notifica que el cierre no es satisfactorio por medio de mensajes y una alarma audible. Este estado termina invariablemente a las 24:00 horas.

Eventos

A continuación se enumeran los diferentes eventos que se suceden en el sistema, y con que acciones se responen a cada uno de ellos:

Inicio del Estado *Standby* Como el nombre indica, este evento sucede cuando el estado del sistema cambia a *standby*, es decir, a las 0:00 hrs.

Se calculan los valores del *Diferencial* en base a las tasas históricas.

Se crea un nuevo archivo de bitácora.

Cleaning Este evento puede dispararse en cualquier momento, siempre y cuando se encuentre dentro del periodo de tiempo del estado *Standby*.

La acción asociada a este evento es eliminar cualquier dato de las páginas de los bancos en las areas destinado a la escritura de cotizaciones.

Inicio del Estado *Round1* Como el nombre indica, sucede cuando el estado del sistema cambia del estado *Standby* al estado *Round1*.

El sistema escribe en la página de alerta un valor que reconoce el sistema de los bancos como inicio de cotización.

Contribution Este evento se dispara cuando un banco ha cotizado valores para la tasa

El sistema obtiene los valores cotizados de la página del banco. Estos valores los registra en el archivo de bitácora. Sólamente si el estado del sistema es *Round1* o *Round2*, almacena los datos en memoria y recalcula los valores de la tasa y las penalizaciones y los muestra en la pantalla.

Reminder Este evento fue creado para avisar a las aplicaciones de los bancos que no han cotizado, que lo hagan.

Ante este evento, el sistema escribe en la página de alerta un valor que las aplicaciones de los bancos reconoce, cuya acción es recordar al usuario final que cotiza. El sistema del banco no notifica al usuario si éste ya ha cotizado.

Inicio del Estado *Round2* Nunca puede dispararse este evento si un número de bancos predefinido ha cotizado ya.

El sistema calcula penalizaciones a aquellos bancos que no cotizaron durante el periodo de tiempo del estado *Round1*.

El sistema escribe en la página de alerta un valor que los programas de los bancos reconocen y que produce que muestre un mensaje indicando que se ha llegado a una segunda ronda.

Inicio del Estado *Final* Este evento sucede sólamente si el estado anterior fue *Round1* o *Round2* y el número mínimo de bancos ha cotizado.

El sistema escribe en la página de alerta un valor que la aplicación de los bancos reconocen, y ante la cual, inhabilitan los campos en donde se escriben los valores de cotización e indica al usuario final que ya no se encuentra habilitada la cotización.

El sistema procede a la publicación de los resultados en las páginas MexIBOR, en el archivo de bitácora y en la base de datos.

Inicio del Estado *Incomplete* Este evento sucede sólamente si el estado anterior fue *round 2*, pero no cotizaron el número mínimo de bancos.

El sistema escribe en la página de alerta un valor que la aplicación de los bancos reconocen, y ante la cual, inhabilitan los campos en donde se escriben los valores de cotización e indica al usuario final que ya no se encuentra habilitada la cotización.

El sistema emite una alarma audible para indicar este estado.

Dado lo anterior, el flujo de operación del sistema MexIBOR es como se ilustra en la figura 2.2.

2.3.2. Operación Manual y Automática

Una premisa importante del diseño es que el sistema debía ser capaz de realizar la operación diaria sin intervención humana. Ésta sólamente sería necesaria para aspectos de cambio de parámetros del mismo, o para situaciones extraordinarias. Sin embargo también debería ser operado en forma manual, para efecto de realización de pruebas.

A efectos de lograr la anterior, en la interfaz del sistema existen controles de selección de estado y de eventos (todos los estados y todos los eventos, a excepción del evento *Contribution*). La selección de estado debería ser mutuamente excluyente, es decir, no permitir la selección de mas de dos estados a la vez, o ninguno de ellos.

La selección de estados debe seguir cierto orden, por ejemplo, no puede seleccionarse el estado *Final* si el actual es *Standby*, o seleccionar el estado *Round2* si el estado actual es *Round1* pero ya han cotizado al número mínimo de bancos.

La selección de eventos también debe seguir ciertas reglas, por ejemplo, no tiene sentido un evento *Reminder* si el estado no es de cotización (*Round1* o *Round2*).

Una vez realizado esto, la operación automática es sencilla: Involucra la creación de un *autómata finito* (que fue denominado *scheduler*), al cual se le programen los tiempos o duraciones de selección de estados y/o eventos, y las condiciones bajo las cuales algunos deben sucederse (por ejemplo el estado *Round2*). A nivel diseño, la operación debería ser idéntica a que este autómata "presionara" los botones a tiempos específicos, dándose cuenta de la secuencialidad, condiciones, etc.

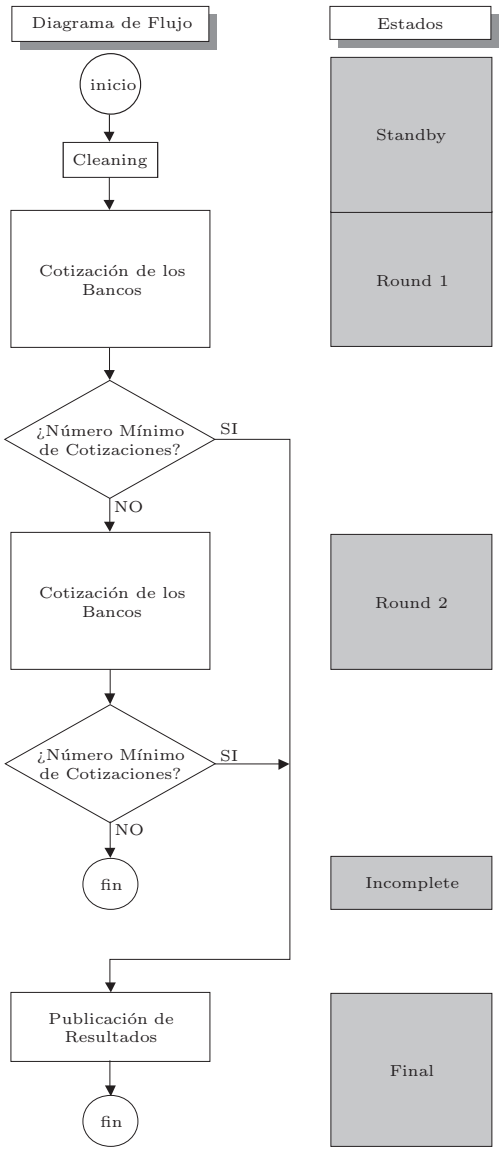

Figura 2.2: Flujo de Operación del Sistema MexIBOR

2.3.3. Separación de Algoritmos de la Implementación Informática

Una parte importante de diseño del sistema MexIBOR es que los algoritmos residieran separados de los programas objeto, en otras palabras, todo el

proceso matemático derivado de los algoritmos no debía estar codificado en los programas fuente.[5]

Esto obedece a las siguientes razones:

1. Que los algoritmos sean fácilmente modificables. Si los algoritmos estuvieran codificados en los programas, la modificación no sería fácil. Para ello sería indispensable utilizar las herramientas de desarrollo utilizadas para los programas (editores de código, compiladores, etc.). Estos algoritmos sólamente podrían ser cambiados por un programador. Amén de que serían poco comprensibles para aquel que no conociera el lenguaje de programación utilizado (dichas fórmulas tendrían que estar convertidas acorde a la sintaxis del lenguaje de programación).

2. Por razones de estética. Se considera una modularización del sistema bastante elegante.

A continuación se detallará el mecanismo informático con el cual se logró esto.

Antes que nada, es necesario darse cuenta que los cuatro algoritmos involucrados §(1.5) no necesariamente deben realizarse al mismo tiempo.

Dado que el diferencial es siempre el mismo para un día cualquiera, el algoritmo del cálculo del diferencial §(1.5.1) sólamente es necesario que se ejecute *una vez al día*.

Para los tres algoritmos restantes, la condición mínima suficiente es que se ejecuten una vez cada día, al tener las cotizaciones definitivas de los bancos, es decir, al inicio del estado *Final*. En el sistema, sin embargo, para efecto únicamente de estética, se ejecutan en cada cotización producida por un banco, a efecto de que en la pantalla de la aplicación se muestre no sólamente la cotización que se realiza en ese momento, sino también los valores de la tasa y penalizaciones que resultan de la nueva situación de valores.

Resultado de lo anterior, se agruparon todos aquellos cálculos que se realizan en tiempos y periodicidades iguales en entidades que fueron denominadas *Scripts*. Un *script* simplemente contiene instrucciones matemáticas que el sistema *Mathematica* entiende.

Como fueron enumeradas tres diferentes tiempos es el sistema, existen tres diferentes *Scripts*, cuyos tiempos de ejecución y algoritmos que implementan son como en el cuadro 2.1.

[5] En el argot informático, se dice *hardcoded*, o vulgarmente en México "hardcodeado", o menos frecuentemente "hardcodificado".

Nombre del *Script*	Periodicidad de ejecución	Algoritmos que contiene
Initialization	Cuando inicia la aplicación	-
Diff	Al inicio de cada día hábil	§1.5.1
Script	En cada cotización	§1.5.2, §1.5.3, y §1.5.4

Cuadro 2.1: Tiempos de Ejecución de *Scripts* en el Sistema MexIBOR

Nótese que como el sistema es diseñado para funcionar de manera contínua, sin intervención humana, el *script Initialization* se ejecuta una vez cada varios días, típicamente semanas o meses (cada vez que el sistema se reinicia), el *script Diff* se ejecuta una vez al día (hábil), mientras que el *script Script*[6] se ejecuta varias veces en un mismo día hábil (típicamente decenas de veces).

Nótese además que el *script Initialization* no contiene o implementa a ninguno de los algoritmos, esto es debido a que sirve para inicializar el entorno *Mathematica* asociado al programa. Como ejemplo mencionaremos el siguiente: Algunos algoritmos (exactamente el algoritmo para el cálculo del *diferencial* §1.5.1) requieren realizar operaciones típicas de estadística, como desviaciones estándar. Cualquier entorno *Mathematica* no contiene por defecto la utilización de operaciones de este tipo, sino que hay que instruír para que utilize un *paquete*, en este caso es un paquete de estadística, de forma tal de que este entorno pueda utilizarlo para operaciones posteriores.

Cada *script* es implementado como un simple archivo de texto. Este contiene instrucciones que reconoce el sistema *Mathematica*. Entre ellas se pueden colocar comentarios (texto que el sistema *Mathematica* ignora) de forma tal que se puede incluír texto que ayude a comprender y/o agrupar las instrucciones contenidas.

El contenido de los scripts, evidentemente debe cumplir con la sintaxis esperada por *Mathemática* en forma de texto. Si bien en el sistema *Mathematica* pueden introducirse expresiones en una forma "natural", por ejemplo:

$$\sum_{i=1}^{max} f \tag{2.1}$$

Siempre se tiene una versión equivalente en texto:

```
Sum[f, {i, max}]
```

[6]El término es desafortunado, posiblemente debió llamarse *Contribution*.

Que es como debe introducirse en los scripts.[7]

La decisión de utilizar archivos de texto es por la facilidad de edición de los mismos (se puede realizar con cualquier editor de texto plano), excluyendo la necesidad de utilizar el editor de comandos de *Mathematica*.

Resultados Numéricos a Cada Paso

Una restricción muy importante en los *scripts* es que todos los resultados que se van obteniendo a cada instrucción que se ejecuta del mismo, el resultado debe ser numérico real o entero[8], o bien puede ser simbólico que puede convertirse a numérico en forma exacta o en el peor de los casos con pérdida de exactitud debido a la truncación o redondeo. Esto se debe a que el programa MexIBOR obtiene los resultados de los cálculos en forma estrictamente numérica.

En el cuadro 2.2 se muestran ejemplos de expresiones, de las cuales algunas de ellas pueden convertirse a su representación real y otras no.

resultado intermedio	representación numérica
5	5
$1/3$	$0.33333\ldots$ (con truncación)
$3+i$	no existe, el resultado no es un número real
x	no existe, el resultado es una literal simbólica
π	$3.141591654\ldots$ (con truncación)
$\sqrt{2}$	$1.4142\ldots$ (con truncación)

Cuadro 2.2: Ejemplos de expresiones y el resultado de intentar convertirlas a valores reales.

La persona que desee leer o modificar los *scripts* debe, por lo tanto tomar en cuenta esta restricción.

Por ejemplo, la expresión:

$$x+1 \tag{2.2}$$

produce un valor numérico sólamente si x es una literal con valor numérico. Cualquier otra cosa (literal no asignada, función, gráfico, vector, matriz, etc.)

[7] Si bien técnicamente funciona el crear las expresiones en modo de comandos, almacenarlo en un archivo y este utilizarlo como *script*, al verlo como texto contiene mucho código extra.

[8] Para excluír otro tipo de "números", como los números complejos.

o inclusive si es una literal no existente en el kernel, provocará que el resultado sea simbólico.

A continuación se presenta el encabezado de un *script*, a modo de mostrar la estructura y documentación de este archivo.

Estructura de los *Scripts*

```
(*****************************************************************************)
(* Script.m                                                                  *)
(* History:                                                                  *)
(* Date                    Explanation & Status                              *)
(* ----------------------- ------------------------------------------------- *)
(* ????????? ?? ???? ??:?? Formula definitions, by Mitch Stonehocker         *)
(* march     21 2000 22:30 Documentation of scripts, by Laurence R Ugalde    *)
                                  :
                                  :
(*****************************************************************************)

(*****************************************************************************)
(* This file is one of three mathematica script files used to calculate      *)
(* the Mexican rate MexIBOR:                                                 *)
(*    - Initialization.m                                                     *)
(*    - Diff.m                                                               *)
(*    - Script.m (this file)                                                 *)
(*****************************************************************************)

(*****************************************************************************)
(* Previous Requirements:                                                    *)
(* ------------------------------------------------------------------------- *)
(* Before any mathematica kernel runs this script, it must have previously   *)
(* run once (and only once) the Initialization.m script, and at least once   *)
(* the Diff.m script (each of these with its own requirements).              *)
(*                                                                           *)
(* Before any mathematica kernel runs this script, it must have been         *)
(* previously instructed to contain the following elements:                  *)
(*    - The AllDataAlg2Matrix variable. It is a matrix of 12 columns by 20   *)
(*      rows, containing the rates submitted by involved banks. Each Row     *)
(*      refers to a particular bank, each Column refers to maturities of     *)
(*      these rates. See section "Logical Order" below.                      *)
(*    - The diff variable. It is an array of 12 elements, containing the     *)
(*      values of diff for each maturity. It can be equal to any diff        *)
(*      previously calculated (algorithm 1), or user defined. See section    *)
(*      "Logical Order" below.                                               *)
                                  :
                                  :
(*****************************************************************************)

(*****************************************************************************)
(* Results:                                                                  *)
(* ------------------------------------------------------------------------- *)
(* After running this script, the mathematica kernel contains the following  *)
(* results of interest:                                                      *)
(*    - The centerRates variable. It is an array of 12 elements containing the*)
(*      center rates, by maturity. See section "Logical Order" below.        *)
(*    - The penalties variable. It is a matrix of 12 columns by 20 rows,     *)
(*      containing the penalties for each bank, for each maturity. Rows      *)
```

```
(*      represent banks, Columns represent  maturities. See section          *)
(*      "Logical Order" below.                                                *)
(*    - The totalPenalties variable. It is an array of 20 elements,          *)
(*      containing the total penalties of each bank (the sum of penalties for *)
(*      each maturity of the bank). See section "Logical Order" below.        *)
(*****************************************************************************)

(*****************************************************************************)
(* Logical Order:                                                            *)
(* ------------------------------------------------------------------------- *)
(* - When a dimension of a uni- or bi-dimensional array (it is: vector of    *)
(*   matrix) represents banks, specifications of which column (or row) of    *)
(*   data is associated to a particular bank is defined by the committee,    *)
(*   and results of interest preserve this order.                            *)
(* - When a dimension of a uni- or bi-dimensional array (it is: vector or    *)
(*   matrix) represents maturities, the number of column/row represents      *)
(*   the maturity in months. i.e. second column/row represents the '2 months' *)
(*   maturity.                                                               *)
(*****************************************************************************)

(*****************************************************************************)
(* Observations:                                                             *)
(* ------------------------------------------------------------------------- *)
(* - All statements of this script must be ended with a semicolon (;). It    *)
(*   is because the intentions of these scripts are to calculate, not        *)
(*   display results, so statements that display or format (for output)      *)
(*   data are not relevant.                                                  *)
(* - Each statement must use elements (symbols, variables, functions, etc.)  *)
(*   previously defined by either mathematica, packages previoulsy loaded,   *)
(*   previous statements in the current script or scripts previously run,    *)
(*   or by previous explicit communication with the mathematica kernel       *)
(*   (by a program, for example). If it were not so, results would not be    *)
(*   numerical, but symbolic.                                                *)
(* - If statements in this script change, please ensure the two previous     *)
(*   specifications be satisfied, also do not delete the statements that     *)
(*   change, instead, put them inside comments, indicating the reason.       *)
(*   New statements must also be documented. Besides an explanation, indicate *)
(*   the date, and update the header of this file.                           *)
(*****************************************************************************)

(*****************************************************************************)
(* Credits:                                                                  *)
(* ------------------------------------------------------------------------- *)
                                  :
                                  :

                  (instrucciones propias del script)

(*****************************************************************************)
```

La parte de interés es el segundo punto de apartado *observaciones*, el cual dice (traducido):

> *Cada instrucción debe usar elementos (símbolos, variables, funciones, etc.) que hayan sido definidos previamente ya sea por Mathematica, paquetes cargados previamente, instrucciones previas en*

el script actual o en scripts ejecutados previamente, o por comunicación explícita con el kernel de Mathematica (por ejemplo, por medio de un programa). De no ser así, los resultados podrían ser no numéricos, sino simbólicos.

2.4. Programación del Sistema MexIBOR

En este apartado se nombrarán aspectos de interés relacionados con el desarrollo del sistema MexIBOR, adicionalmente a los ya mencionados.

El sistema fue escrito en lenguaje C++ *(Microsoft Visual C++)* para ser ejecutado en un entorno *Windows*. Inicialmente se iba a utilizar el lenguaje *Java* para la programación del sistema, desafortunadamente, el sistema *JLink* que es la interfaz de *Mathematica* para *Java*, se encontraba en versión alfa. Asimismo se encontraban en primeras versiones la interfaces para *Java* de las tecnologías *REUTERS*. Es una lástima, puesto que el sistema hubiera sido mucho mas portable y por lo tanto mas estable frente a los cambios de versiones de sistemas operativos y herramientas.

2.4.1. Multisubproceso *(Multithreading)*

En el sistema MexIBOR siempre ejecutan al menos dos *threads*: el *thread* principal (el que responde a eventos provenientes de la interfaz (cuando se presiona un botón, o se selecciona un menú) y un *thread* secundario, creado a partir del principal en los primeros momentos de ejecución del sistema. Este *thread* secundario verifica a cada segundo, si ha llegado la hora de iniciar un nuevo estado o de lanzar un nuevo evento, y está relacionado directamente con la parte del sistema denominado *scheduler* (por ejemplo el evento *Cleaning*, que se suceden a horas fijas o preprogramadas).

Si bien existen dos *threads* propios del sistema, existe otro *thread* que si bien no pertenece al sistema (en el sentido de que el sistema no lo crea), si afecta al sistema. Estos son los *threads* que ejecutan funciones *callback* del sistema. Una función *callback* es un segmento de código que un programa concede a otro programa, de forma tal que este último (programa) pueda ejecutarlo cuando así decida hacerlo. Generalmente son utilizadas cuando el primer programa responde a eventos que controla el segundo programa. En el caso MexIBOR este programa externo es un programa de la infraestructura *REUTERS* que avisa al sistema MexIBOR de cambios en las páginas *REUTERS* involucradas en el sistema, por ejemplo, cuando por medio de la aplicación del banco se

escriben cotizaciones en la página (del banco), el sistema *REUTERS* lo detecta y avisa al sistema MexIBOR, obviamente esprecificando los detalles del cambio en la página (cuál página, que parte de la página y los valores).

Los programas multisubproceso *(multithreading)* nunca son sencillos de escribir, y resultan necesarios mecanismos de sincronización de procesos y/o subprocesos.

En el caso del sistema MexIBOR siempre se tienen dos o tres subprocesos *(threads)* ejecutando concurrentemente diversas partes del código.

Capítulo 3

Aritmética Matemática y Computacional

3.1. Números Enteros

Sabemos que la cantidad de números naturales es infinita, es decir, que si comenzamos a contar $0, 1, 2, \ldots$ no terminaríamos nunca. Los números naturales (representados por $\mathbb{N}$) son los primeros que aprendemos por el acto natural de contar. El siguiente paso en nuestra vida escolar seguramente son los números enteros (representados por $\mathbb{Z}$), que comprenden los números naturales y además los números negativos y el cero. En ese momento nos damos cuenta que la anterior infinitud no sólamente existe hacia la dirección del eje positivo, sino además hacia el eje negativo de la escala numérica. Este nivel de infinitud fue definido por Cantor[1] como $\aleph_0$ (aleph cero) y representa lo que es contable, o que puede ser correspondido biunívocamente con el conjunto infinito de los números enteros.

Siempre que deseemos expresar una cantidad en forma numérica exacta nos damos cuenta que el espacio que necesitamos para hacerlo depende de la *magnitud* del número, La magnitud puede ser definida como la distancia del número desde el cero, por lo tanto los números -123 y 123 tienen la misma magnitud, sin importar que $-123 < 123$. De esta misma manera, -500 tiene mayor magnitud que 100, aunque 100 tenga mayor valor numérico que -500. Otra manera de expresar la magnitud de un número es haciéndo uso del *valor absoluto*, cuyo valor se define como:

[1] Georg Ferdinand Ludwig Philipp Cantor.

$$|x| = \begin{cases} x & x \geq 0 \\ -x & x < 0 \end{cases} \tag{3.1}$$

En nuestra acostumbrada notación decimal, el espacio necesario (definido en número de cifras) para expresar un número n es:

$$cifras(n) = \lfloor \log_{10} |n| \rfloor + 1 \tag{3.2}$$

Esto es una consecuencia inmediata de la naturaleza exponencial de la notación posicional.

En forma inversa, la cantidad de posibles números que podemos representar si tenemos espacio para n cifras es de 10^n.

En las computadoras, cuyo sistema de numeración es binario (base 2) han existido varias formas de representación de valores numéricos enteros, aunque la mas utilizada es la posicional (igual que nosotros) por tener las ventajas de que se pueden realizar operaciones aritméticas con mucha facilidad (al igual que nuestro sistema posicional decimal). En sistemas de representación no posicional (por ejemplo EBCDC) las operaciones aritméticas, aún las mas básicas se vuelven complicadas.[2]

De forma análoga, la cantidad de dígitos binarios (bits) necesarios para expresar el número n es:

$$cifras(n) = \lfloor \lg |n| \rfloor + 1 \tag{3.3}$$

En dónde $\lg x = \log_2 x$. Algunas veces se utiliza un bit mas para determinar el signo del número, Asimismo, si contamos con n bites disponibles, podemos representar una cantidad de posible números binarios de 2^n.

La agrupación mínima de bits generalmente utilizada es de 8 bits, denominada un byte. De acuerdo a lo anterior, la cantidad de números binarios que se pueden representar son $2^8 = 256$ en el rango de 0 al 255. Si se desea utilizar un bit para el signo entonces disminuye la cantidad de bits para representar la magnitud a $2^7 = 128$, lo cual queda un rango desde -128 a 127.

El número usual de bytes es generalmente de potencias de dos, como se ejemplifica en el cuadro 3.1.

La mayoría de las computadoras soportan estos esquemas de representación y cálculos aritméticos de números enteros de forma intrínseca como operaciones habituales de sus procesadores.

[2] ¿Alguna vez el lector ha imaginado como resolver una simple suma entre dos números expresados en números romanos, cuya notación no es posicional?

3.2. Números Reales

Durante nuestra formación escolar, nos damos cuenta de que además de que la línea (punteada) de los números enteros es infinita, existe una cantidad infinita de números entre cada número entero. La línea no es punteada, sino contínua: Los números *reales* (representados como $\mathbb{R}$). Ante esta nueva *dimensión* de infinitud al conjunto de los números reales Cantor asoció en consecuencia con $\aleph_1$ (aleph uno) a aquellos conjuntos que pueden hacerse corresponder biunívocamente con el conjunto de los números reales.

En nuestra forma de representación decimal, se utiliza el punto para delimitar la parte entera de la parte fraccionaria. Afortunadamente, la notación posicional permite utilizar exactamente el mismo esquema, utilizando simplemente exponentes negativos para la parte fraccionaria, por lo que los métodos usuales de operaciones aritméticas manuales son prácticamente los mismos que los utilizados para números enteros.

En la computadora puede seguirse este mismo principio, simplemente reservando una cantidad de bits para la parte entera y otra para la parte fraccionaria (y un bit de signo, si se pretenden utilizar cantidades negativas). Por ejemplo, supongamos que disponemos de dos bytes (16 bits) para representar números reales, y decidimos utilizar un bit para el signo, 7 bits para la parte entera y los restantes 8 bits para la parte fraccionaria. La parte fraccionaria nos dice vamos a dividir la unidad en $2^8 = 256$ partes. Todos los números que podemos representar están en el rango $-128\frac{255}{256} \ldots 127\frac{255}{256}$ a intervalos de $\frac{1}{256}$. Este esquema de representación de números reales es comúnmente conocido como de *punto fijo*, o *punto virtual*. Generalmente no forma parte intrínseca de las operaciones de los procesadores en equipos comerciales. Es sin embargo comúnmente implementada en manejadores de bases de datos.

La forma de representación comúmnente implementada en los procesadores

número de bytes B	número de bits $b = 8B$	número de posibles valores 2^b
1	8	256
2	16	65536
4	32	4294967296
8	64	18446744073709551616

Cuadro 3.1: Número de valores enteros que pueden ser representados utilizando agrupaciones comúnes de bytes.

es la llamada de *punto flotante*. Este formato hace uso de lo que se define como *número de cifras significativas*, según lo cual, el valor informativo de un número se encuentra en las últimas cifras (de izquierda a derecha), sin importar su magnitud (la misma que se analizaba en la subsección anterior, es decir, su *distancia* desde el origen). Así, en números grandes, la parte fraccionaria e incluso algunas de las primeras (de izquierda a derecha) de la parte entera, son de poca información, porque se pueden suponer como cero. Esto se puede aclarar mejor con ejemplos. El número 12345.67 con tres cifras significativas es 12300, con una cifra significativa es 10000, con seis cifras significativas es 12345.6 y con diez es 12345.67000 (aunque tiene el mismo valor numérico que el original). Evidentemente que el número será mas exacto (mas cercano al número que queramos representar) mientras mas cifras significativas tomemos.

En este formato se almacenan el signo, el valor de las cifras significativas, denominado la *mantisa*, y la parte que indica su magnitud, denominado el *exponente*. La equivalencia del "mundo real" que mas se aproximaría a este formato sería la *notación exponencial*, en la cual todo número puede ser expresado como un valor de mantisa entre 0 y 10 (no incluído éste) y valor exponente, de la forma $mantisa \times 10^{exponente}$, como en los mostrados en el cuadro 3.2.

número	número expresado en notación exponencial
0	0×10^{0}
3.14	3.14×10^{0}
50	5×10^{1}
50.001	5.001×10^{1}
0.5	5×10^{-1}
10	1×10^{1}
12345	1.2345×10^{4}
.00012345	1.2345×10^{-4}

Cuadro 3.2: Ejemplos de números expresados en notación científica.

Hay varias maneras de representar un mismo múmero en notación exponencial, por ejemplo el número 50 puede ser representado como 50×10^{0}, 5×10^{-1}, 50000×10^{-3} o $0.000000005 \times 10^{10}$. Se dice que un número en notación exponencial se encuentra *normalizado* si la mantisa se encuentra en forma de número real con parte entera de un sólo dígito no cero[3]. De esta forma el número 50 en notación exponencial normalizada es 5×10^{1}, y 315 es 3.15×10^{2}.

[3]La excepción es el mismo número cero, cuyo formato en notación exponencial normali-

En este formato, la mantisa proporciona la información *significativa* del número, y el exponente la información acerca de su *magnitud*. Un estándar común para este formato de representación es el denominado IEEE 754.

3.3. Problemática de la Aritmética Computacional

Las operaciones aritméticas usuales en los números enteros y reales gozan de algunas propiedades. La que vamos a resaltar es la propiedad de clausura, la cual significa que una operación efectuada entre dos elementos del conjunto produce un resultado que se encuentra en el mismo conjunto. En el cuadro 3.3 se especifica cuales operaciones aritméticas presentan esta propiedad.

operación	conjunto	propiedad de clausura
adición	$\mathbb{Z}$	si
adición	$\mathbb{R}$	si
sustracción	$\mathbb{Z}$	si
sustracción	$\mathbb{R}$	si
multiplicación	$\mathbb{Z}$	si
multiplicación	$\mathbb{R}$	si
división	$\mathbb{Z}$	no
división	$\mathbb{R}$	si

Cuadro 3.3: Propiedad de clausura de operaciones aritméticas en el conjunto de números enteros y reales.

Como puede verse, todas estas operaciones presentan la propiedad de clausura cuando operan en el conjunto de los números reales. En el conjunto de los enteros no lo presenta la división, ya que al dividir un número entero por otro, el resultado no necesariamente es entero.[4]

La mayoría de los errores derivados de la aritmética computacional provienen del hecho de que dichas propiedades de clausura no se preservan en la aritmética computacional. Esto es derivado del hecho de que en la aritmética

zada es 0×10^0

[4]La división de números enteros sólamente produce resultados enteros si el dividendo es múltiplo de divisor.

matemática la cantidad de números es infinita y en la aritmética computacional no.

Las computadoras generalmente presentan diferentes comportamientos ante este problema dependiendo si se utiliza aritmética entera o real.

3.3.1. Aritmética Entera Computacional

El problema surge cuande se realiza una operación aritmética en cuyo resultado la magnitud es superior a la máxima magnitud representable por la computadora. Dado lo visto al inicio del capítulo (§3.1), esta magnitud depende de la cantidad de memoria (bytes) utilizados para representar números enteros. Esta situación recibe el nombre de desbordamiento *(overflow)*.

Las operaciones que realiza la computadora internamente al realizar operaciones aritméticas de número enteros, es la denominada *Aritmética Modular*.

En dicha aritmética, que es la utilizada en la teoría de números[5], se utilizan las mismas operaciones aritméticas comunes que en la aritmética entera convencional, pero a diferencia de ésta, el conjunto de números en que opera es finito y presenta propiedades de clausura.

En la aritmética modular (dado un número n), el conjunto de números enteros es finito y contiene n elementos, 0, 1, ... $(n-1)$ y es representado como $\mathbb{Z}_n$. El número siguiente al número $n-1$ es nuevamente el número 0. Los resultados de las operaciones aritméticas son equivalentes al resultado convencional, el cual se ha dividido entre el módulo y obtenido su residuo, es decir, como si los resultados siempre fueran los residuos, como se ejemplifica en el cuadro 3.4.

Módulo	Operación	Resultado	Explicación
$\mathbb{Z}_{10}$	5×3	5	$15 \equiv 5 \mod 10$
$\mathbb{Z}_{20}$	$15 + 15$	10	$30 \equiv 10 \mod 20$
$\mathbb{Z}_{256}$	$200 + 100$	44	$300 \equiv 44 \mod 256$

Cuadro 3.4: Ejemplos de operaciones de aritmética modular.

De acuerdo con la anterior, la computadora, al utilizar aritmética modular, no distingue entre las situaciones de desbordamiento *(overflow)* y las *normales*,

[5]Que desafortunamanente no se instruye en la formación escolar, por considerarse hasta hace poco tiempo inútil, sin embargo es fundamental para las ciencias de la información y criptografía.

ámbas son correctas (dentro de la teoría de la aritmética modular), y por lo tanto indistinguibles. Es muy conocido el hecho de que no es posible detectar el desbordamiento *(overflow)* en la aritmética entera computacional común, ya que este simplemente no existe.

Esto nos lleva a la sorprendente conclusión, que si utilizamos tipos de datos numéricos enteros de un *byte* (256 valores) e intentamos realizar la operación $200 + 100$ (véase el ejemplo del cuadro 3.4), el resultado será de 44 (lo cual sabemos que es un desbordamiento) y además no va a ser posible detectarlo.[6]

3.3.2. Aritmética Real Computacional

De acuerdo a lo visto en el apartado de números reales (§3.2), se acostumbra representar a estos números por medio de la combinación significatividad-magnitud.

Dado que la magnitud es un valor exponencial, no sólamente puede utilizarse para representar valores grandes (magnitud grande positiva), sino también números muy pequeños (magnitud grande negativa). El desbordamiento se presenta si el número es mas grande que el mayor número representable. Cuando el número es muy pequeño (muy cercano a cero), menor al menor número representable se denomina nulificación *(underflow)*[7]. A diferencia de la aritmética entera, en la aritmética real (computacionales) generalmente si se cuenta con mecanismos intrínsecos para detectar el desbordamiento *(overflow)* y la nulificación *(underflow)*.

Sin embargo, hay otra problemática de la computación real. Como se vió en la misma sección §3.2, los números reales al igual que los números enteros tienen una dimensión de infinitud de extensión. Los números reales tienen una dimensión de infinitud adicional: Entre cada par de números enteros existe una cantidad infinita de números reales. Evidentemente no todos esos números pueden ser representados con una cantidad de memoria finita. Por lo tanto, la aritmética real computacional, además cuenta con este problema. La distancia que existe entre dos números representables adyacentes se denomina *ULP (Unit in the Last Place)* y no siempre es un valor constante, sino que depende de la magnitud. En números con magnitud grande, el valor del *ULP* es mayor que entre números con magnitud pequeña.

Cuando se presenta una operación entre números reales cuyo resultado no se encuentra dentro del conjunto representable, se tiene que decidir hacia cuál

[6]Por medios intrínsecos a la computadora, o al lenguaje de programación. Si es posible detectarlo, pero el programador debe implementar los medios para ello.

[7]No existe una traducción literal satisfactoria.

de los dos números adyacentes representables tiene que ajustarse, existiendo por lo tanto un error máximo de $\pm\frac{1}{2}ULP$.

3.4. Aritmética Racional

3.4.1. Números Racionales

Todos los números enteros, los no enteros cuya representación exacta tiene parte fraccionaria de tamaño finito, o que es infinita pero formada por un grupo (de cualquier tamaño) de cifras que se repiten indefinidamente, puede ser expresado en forma *racional*.

Los números racionales (simbolizados por $\mathbb{Q}$) son definidos como:

$$\mathbb{Q} = \left\{ \frac{a}{b} \mid a, b \in \mathbb{Z}, b \neq 0 \right\} \tag{3.4}$$

Es decir, los números racionales son los representados en forma de fracción cuyos numerador y denominador son números enteros. En el cuadro 3.5 se ejemplifican varios números convertidos a su representación racional.

Número	Número expresado en forma racional
8	8/1
3.5	7/2
$5.333\ldots = 5.\overline{3}$	16/3
$1.142814281428\ldots = 1.\overline{1428}$	8/7
$-0.08333\ldots = -0.08\overline{3}$	−1/12

Cuadro 3.5: Ejemplos de números expresados en forma racional

La aritmética de números racionales es bien recordada (no a todos gratamente), en nuestras clases de instrucción primaria esta es conocida como los "quebrados".

Las computadoras en forma general no presentan soporte intrínseco a la manipulación aritmética racional (por ejemplo, como instrucciones propias de microprocesador). Sin embargo, una implementación resulta ser sencilla (en §5 se muestran los principios para ello).

Para recordar, enunciaremos dichas reglas aritméticas de números racionales en el cuadro 3.6.

3.4.2. Correspondencia entre Números Racionales y Reales

Todos sabemos que el número real equivalente a un número racional siempre va a ser de longitud finita, o de longitud infinita pero que consta de un grupo de dígitos que se repite indefinidamente.

Lo que no es muy común saber es que existe un regla que determina si el número real correspondiente a cierto número racional será de tamaño finito o infinitamente periódico. Mucho menos conocido es el hecho que además dicha regla depende de la base de numeración utilizada para representar dicho número. Esto lleva a la sorprendente conclusión que un mismo número racional (sin importar en que base de numeración sea representado), su correspondiente número real puede ser de tamaño finito en cierta base, e infinitamente periódico en otra.

Por simplicidad, utilicemos números racionales de la forma $1/x$.

La regla es la siguiente: Sean a_1, a_2, ..., a_n los n factores primos unicos de la base de numeración b. Entonces todos los números racionales de la forma:

$$\frac{1}{a_1^{x_1} a_2^{x_2} \dots a_n^{x_n}} \tag{3.5}$$

para cualesquiera valores $x_1, x_2, \dots, x_n$ enteros positivos o cero, tendrán representación real finita en la base de numeración b, Los números que no pueden ser obtenidos de tal forma, no representación finita, sino infinitamente periódica.

Para nuestra base de numeración 10 (factores primos 2 y 5), todos los números racionales de la forma

$$\frac{1}{2^x 5^y} \tag{3.6}$$

operación	regla
adición $\frac{a}{b} + \frac{c}{d}$	$\frac{ad+cb}{bd}$
sustracción $\frac{a}{b} - \frac{c}{d}$	$\frac{ad-cb}{bd}$
multiplicación $\frac{a}{b}\frac{c}{d}$	$\frac{ac}{bd}$
división $\frac{a/b}{c/d}$	$\frac{ad}{bc}$

Cuadro 3.6: Reglas de aritmética de números racionales.

Para cualquier valor positivo entero de x y y serán de longitud finita. En el cuadro 3.7 se muestran ejemplos de ello.

Número racional	Regla	Representación real
1/2	$2^1 5^0$	0.5
1/4	$2^2 5^0$	0.25
1/5	$2^0 5^1$	0.2
1/8	$2^3 5^0$	0.125
1/10	$2^1 5^1$	0.1
1/16	$2^4 5^0$	0.0625
1/20	$2^2 5^1$	0.05
1/25	$2^0 5^2$	0.04
1/32	$2^5 5^0$	0.03125
1/40	$2^3 5^1$	0.025
1/50	$2^1 5^2$	0.02
1/64	$2^6 5^0$	0.015625
1/80	$2^4 5^1$	0.0125
1/100	$2^2 5^2$	0.01

Cuadro 3.7: Ejemplos de racionales con representación real finita en base 10.

Ahora analizaremos la base de numeración 2. Su único factor primo es él mismo, por lo tanto todos los números racionales de la forma:

$$\frac{1}{2^x} \tag{3.7}$$

Serán de tamaño finito al ser representados como números reales *en base 2*. En el cuadro 3.8 se muestran lo ejemplos para esta base.

Número racional	Regla	Representación real (en binario)
1/2	2^1	0.1_2
1/4	2^2	0.01_2
1/8	2^3	0.001_2
1/16	2^4	0.0001_2
1/32	2^5	0.00001_2
1/64	2^6	0.000001_2

Cuadro 3.8: Ejemplos de racionales con representación real finita en base 2.

Para dar ejemplos de números con representación real finita en una base, e infinitamente periódica en otra, el número racional 1/5 tiene una representación real en decimal finita (0,2), mientras que en binario es infinitamente periódica ($0.001100110011\ldots_2 = 0.\overline{0011}_2$).

Nótese además, que un número racional $1/x$ cuya representación real en binario es siempre finita, *siempre* será también finita en decimal, pero esto se debe el hecho de que cualquier número de la forma 2^x puede verse como (y es equivalente a) $2^x 5^0$, esto es: Un número que en base decimal tiene representación real de longitud finita no necesariamente será de representación finita en base binaria, pero lo contrario si es cierto: si un número expresado en base binaria tiene representación real finita, también la tendrá en decimal.

3.5. Números Irracionales

Los números irracionales son aquellos cuya representación tiene una parte fraccionaria de longitud infinta, y que no presenta grupos cíclicos, por lo que no pueden ser representados en forma racional.

Algunos de estos números irracionales, sin embargo, si pueden ser representados como un conjunto finito de operaciones aritméticas y operaciones de raíces (cuadradas o de otro radical) entre números enteros o racionales. Estos números son denominados *no trascendentes*. Dicho formalmente, son números que son raíces de alguna ecuación algebraica de coeficientes enteros o racionales. Por ejemplo, la longitud de la diagonal de un cuadrado cuyos lados miden una unidad, es una cantidad que no puede ser expresada en forma racional, pero si como un conjunto de operaciones algebraicas. Exactamente equivale a $\sqrt{2}$. Equivalentemente, es una solución de la ecuación algebraica $x^2 = 2$. Otro número de este tipo es la denominada razón aúrea, que es expresado algebraicamente como $(\sqrt{5}-1)/2$, y es solución de la ecuación algebraica $x^2 - x = 1$.

Cuando un número fraccionario no puede ser representado en forma exacta, ni racional o como una expresión algebraica finita, se dice que es un número *trascendental*. Existen muchos de ellos de gran importancia debido a las propiedades que en el mundo real y matemático presentan, pero debido a la imposibilidad de representación exacta, racional o algebraica, se les asignan símbolos bien conocidos. Posiblemente el mas conocido de ellos sea la razón entre la circunferencia y del diámetro de cualquier círculo, cuyo símbolo es π.

3.6. Errores de Interpretación Humana

Un error muy común es creer que al truncar o redondear información de forma manual o automática, la exactitud prevalece. Por ejemplo supongamos que deseamos tabular proporciones como porcentajes, primero truncando los resultados para obtener porcentajes sin decimales como en el cuadro 3.9, o redondeando como en el cuadro 3.10. Nótese como en la suma de proporciones truncadas o redondeadas no se obtiene el 100 %.

Dato	Incidencias	Proporción Real	Truncado
Dato 1	333	$333/100 = 0.333$	33 %
Dato 2	501	$501/100 = 0.501$	50 %
Dato 3	166	$166/100 = 0.166$	16 %
TOTAL	1000	1	99 %

Cuadro 3.9: Ejemplo de pérdida de información por truncación.

Dato	Incidencias	Proporción Real	Redondeado
Dato 1	268	$268/100 = 0.268$	27 %
Dato 2	566	$566/100 = 0.566$	57 %
Dato 3	166	$166/100 = 0.166$	17 %
TOTAL	1000	1	101 %

Cuadro 3.10: Ejemplo de alteración de información por redondeo.

Capítulo 4

Problemática Detectada en el Sistema MexIBOR y Solución

4.1. Problemática

Primero se hará el ejercicio de calcular el valor de la tasa y penalizaciones, dados los valores de cotizaciones de ejemplo y un valor para el diferencial previamente calculado en 0.025.

Suponga que los bancos han cotizado los valores que se muestran en el cuadro 4.1.

banco	cotización
$banco_1$	16
$banco_2$	16
$banco_3$	16.05
$banco_4$	15.5
$banco_5$	15.7
$banco_6$	16.1
$banco_7$	16
$banco_8$	17.25

Cuadro 4.1: Cotizaciones de ejemplo.

De acuerdo con el algoritmo de cálculo de la tasa (§1.5.2), se realizan los siguientes cálculos (los elementos de los vectores son mostrados en forma vertical):

$\overrightarrow{Xa}$	$\overrightarrow{Xp}$	$\overrightarrow{Xa'}$ $= \overrightarrow{Xa} + 0.025$	$\overrightarrow{Xp'}$ $= \overrightarrow{Xp} - 0.025$	$\overrightarrow{Xp'}$ - $\overrightarrow{Xa'}$
15.5	17.25	15.525	17.225	1.7
15.7	16.1	15.725	16.075	0.35
16	16.05	16.025	16.025	0
16	16	16.025	15.975	-0.05
16	16	16.025	15.975	-0.05
16.05	16	16.075	15.975	-0.1
16.1	15.7	16.125	15.675	-0.45
17.25	15.5	17.275	15.475	-1.8

Cuadro 4.2: Cálculos intermedios para el cálculo de la tasa con los valores de ejemplo.

De acuerdo con este mismo algoritmo, se define el valor de u como el número de componentes positivos de el último vector $\overrightarrow{Xp'} - \overrightarrow{Xa'}$, que es de 2 (los dos primeros).

Los valores para r_1 (ecuación 1.10), r_2 (ecuación 1.11) y el valor de la tasa (ecuación 1.12) son:

$$\begin{aligned} r_1 &= max\{Xa'_u, Xp'_{u+1}\} \\ &= max\{Xa'_2, Xp'_3\} \\ &= max\{15.725, 16.025\} \\ &= 16.025 \end{aligned} \tag{4.1}$$

$$\begin{aligned} r_2 &= min\{Xa'_{u+1}, Xp'_u\} \\ &= min\{Xa'_3, Xp'_2\} \\ &= min\{16.025, 16.075\} \\ &= 16.025 \end{aligned} \tag{4.2}$$

$$T = \frac{r_1 + r_2}{2} = 16.025 \tag{4.3}$$

De acuerdo con el algoritmo de identificación de cotizaciones fuera de rango (§1.5.3), todas las cotizaciones que se encuentren en el rango 16.025 ± 0.025,

o equivalentemente, dentro del intervalo (cerrado) [16, 16.05] se consideran como dentro del rango de tasas mercado, y por lo tanto no tienen penalización alguna. De acuerdo con el algoritmo para el cálculo de penalizaciones (§1.5.4), el resto de cotizaciones (las que se encuentran fuera del rango o intervalo) se les calcula una penalización que tiene un valor mínimo, es decir, una penalización que nunca puede ser menor a determinado valor.

En el cuadro 4.3 se muestra, de acuerdo con los resultados anteriores, cuales bancos hicieron cotizaciones dentro del rango de tasas de mercado:

banco	cotización	en rango de tasas de mercado
$banco_1$	16	si
$banco_2$	16	si
$banco_3$	16.05	si
$banco_4$	15.5	no
$banco_5$	15.7	no
$banco_6$	16.1	no
$banco_7$	16	si
$banco_8$	17.25	no

Cuadro 4.3: Cotizaciones en rango de tasas de mercado, de acuerdo con los valores y cálculos del ejemplo

Sin embargo, al utilizar estos mismos datos como prueba en el sistema, se detectó que la cotización del $banco_3$ se encontraba fuera del rango de tasas de mercado, y por lo tanto con una penalización (la penalización mínima). Se verificó desde luego, que no hubiera un error en los *scripts*, en el programa o en los datos introducidos.

Siendo todos los elementos exhaustivamente verificados, se procedió a hacer el cálculo utilizando *Mathematica* en modo de comando, cada uno de los pasos y con los mismos datos.

Específicamente, en la determinación del valor del tercer componente del vector $\overrightarrow{Xp'} - \overrightarrow{Xa'}$, (véase el cuadro 4.2, en el tercer renglón), implica el cálculo de $(16.05 - 0.025) - (16 + 0.025)$, cuyo resultado exacto (como se muestra en el cuadro) es de 0. Al momento de hacer los cálculos manuales en *Mathematica* obtenemos los resultados de la figura 4.1.

A primera vista, pareciera un error de *Mathematica*, sin embargo, se hizo el mismo ejercicio utilizando el lenguaje de programación *Java*, obteniéndose los resultados mostrados en la figura 4.2.

```
In[1]:=   x = 16 + 0.025  ↩
Out[1]:=  16.025

In[2]:=   y = 16.05 - 0.025  ↩
Out[2]:=  16.025

In[3]:=   y - x  ↩
Out[3]:=  3.55271 x 10^-15
```

Figura 4.1: Cálculos erróneos, obtenidos con *Mathematica*.

```
double x = 16 + 0.025;
double y = 16.05 - 0.025;
System.out.println(y - x);

salida:
3.552713678800501E-15
```

Figura 4.2: Mismos cálculos, utilizando el lenguaje *Java*.

También se probó con el lenguaje *C/C++*, y el sistema *AWK*, obteniéndo idénticos resultados, por lo que la posibilidad de un *bug* en el sistema *Mathematica* queda excluída.

La particularidad de los programas escritos en *Java* es que *siempre* utilizan el mismo formato de representación de números de punto flotante, que es precisamente el IEEE 754, sin importar si el equipo en donde se ejecuta el programa tiene soporte intrínseco para ello o no. Asimismo, al consultar la documentación del sistema *Mathematica* se observa que este sistema trata por defecto todos los números con parte fraccionaria de acuerdo con el formato de representación de punto flotante intrínseco de la máquina. En la máquina en donde se realizó el ejercicio incorpora también el formato IEEE 754, por lo que se llega a los mismos resultados.

El formato IEEE 754 (de precisión doble) es simplemente la representación exponencial normalizada de valores numéricos, evidentemente expresados en base 2, en lugar de base 10, con una longitud de mantisa de 52 bits, un valor para el exponente de 11 bits (valor con signo) y un bit para el signo del número (un total de 64 bits, u 8 bytes). Nótese que no es necesario reservar un bit para la parte entera de la mantisa, puesto que para que sea normalizado la parte

entera debe ser de un sólo dígito y no ser cero, pero puesto que en sistema binario sólamente hay un dígito que no es cero (el uno), se puede asumir que siempre es uno[1] y utilizar los 52 bits sólamente para representar la parte fraccionaria de la mantisa.

Se hablará en este apartado, por lo tanto, de algunos aspectos importantes de la representación IEEE 754, sin embargo esta discusión no será exhaustiva[2], sino sólamente se discutirán aquellos elementos relevantes a la problemátiva presentada en el cálculo de la tasa MexIBOR.

De acuerdo a lo anterior, el número 16 es representado en el formato IEEE como:

$$16 = 1.0000000000000000000000000000000000000000000000000000_2 \times 2^4 \tag{4.4}$$

Nótese que el subíndice 2 de la mantisa indica que el número está expresado en base 2. Véase también que la base y el exponente están expresados en base 10 (sin subíndice). Esto se ha hecho deliberadamente porque para nuestro ejercicio es necesario ver la mantisa como *bits*, es decir en base 2, pero si nos interesa la magnitud y valor (para los humanos) del exponente, por lo que es mas conveniente que sea expresado en decimal.[3]

Es de notar además que la mantisa se encuentra normalizada y que se utilizan 52 dígitos binarios para su parte fraccionaria.

Al momento de representar el número 0.025 (parte fraccionaria finita) nos damos cuenta que este número, en binario tiene una parte fraccionaria de longitud infinita! (por lo visto en la sección §3.4.2). El número en sí no es irracional, por lo que su parte fraccionaria esta conformada por grupos de cifras que se repiten indefinidamente. En el caso de este número:

$$0.025 = 0.00000110011001100\ldots_2 = 0.00000\overline{1100}_2 \tag{4.5}$$

Evidentemente que en formato IEEE 754, este valor tiene que estar normalizado y además truncar a los primeros 53 dígitos binarios, por lo tanto 0.025 expresado en formato IEEE 754 es:

[1] Este es denominado *bit "oculto"*.

[2] El estándar IEEE 754 es muy amplio, abarca inclusive el manejo y representación de infinitos positivos y negativos, *underflow* gradual, ceros positivos y negativos, resultados de operaciones no definidas (como división por cero). Mas allá de la representación, define además la mecánica de las operaciones aritméticas precisas entre números en este formato.

[3] Este mismo número expresado totalmente en binario sería $1.0000000000000000000000000000000000000000000000000000 \times 10^{100}$, un "*googol*" binario.

$$0.025 = 1.10011001100110011001100110011001100110011001100110011010_2 \times 2^{-6} \tag{4.6}$$

La parte "final" de la mantisa no parece obedecer el patron del grupo repetitivo, sin embargo es correcto debido al principio del formato IEEE 754 que dictamina que si el número a representar no se encuentra dentro del conjunto de valores IEEE 754 (este es el caso, puesto que el valor real no se encuentra debido a que es de longitud infinita), el valor IEEE 754 a utilizar es aquel que se encuentre mas cercano (en distancia) al valor real.

El valor IEEE 754 que se encuentra inmediatamente a la izquierda del valor real de 0.025 (al cual denominaremos 0.054^- es exactamente el valor con los bits *truncados* y equivale exactamente a:

$$\begin{aligned} & 0.025^- \\ &= 1.10011001100110011001100110011001100110011001100110011001_2 \times 2^{-6} \\ &= 0.02499999999999999791833182882783148670569062232971191406 25 \end{aligned} \tag{4.7}$$

La distancia de este número con el valor real 0.025 es:

$$|0.025 - 0.025^-| = 2.0816681711721685132943093776702880859375 \times 10^{-18} \tag{4.8}$$

El valor siguiente representable por el formato IEEE 754 a la derecha del valor real 0.025, que denominaremos 0.025^+ es:

$$\begin{aligned} & 0.025^+ \\ &= 1.10011001100110011001100110011001100110011001100110011010_2 \times 2^{-6} \\ &= 0.02500000000000000013877787807814456755295395851135253906 25 \end{aligned} \tag{4.9}$$

La distancia de este número con el valor real 0.025 es:

$$|0.025 - 0.025^+| = 1.3877787807814456755295395851135253906 25 \times 10^{-18} \tag{4.10}$$

Comparando sus "distancias", el valor 0.025^+ es mas cercano al valor real 0.025:

$$|0.025 - 0.025^{-}| > |0.025 - 0.025^{+}| \tag{4.11}$$

Al momento de hacer una suma (o resta), el operando con exponente menor debe convertirse al equivalente con el mismo exponente que el operando con exponente mayor. En el ejemplo, la representación del número 6 tiene un exponente de 4, mientras que la representación de 0.025 tiene un exponente de -6. Esto se hace haciendo un corrimiento de bits a la derecha tantas posiciones como sean necesarias, insertando ceros a la izquierda y descartando los bits que se corren a la derecha. Esto desde luego, provoca la pérdida de estos bits y consecuentemente la pérdida de información. El número de posiciones a hacer el corrimiento es de $4 - (-6) = 10$, por lo que el número queda como:

$$0.025 = 0.0000000001100110011001100110011001100110011001100110_2 \times 2^4 \tag{4.12}$$

Nótese que el número queda temporalmente no normalizado, a efectos de poder realizar la operación aritmética.

Una vez que ambos números se encuentran con el mismo exponente, realizar la suma es simplemente una operación de suma de bits:

$$\begin{array}{rl} & 1.00000000000000000000000000000000000000000000000000_2 \times 2^4 \\ + & 0.0000000001100110011001100110011001100110011001100110_2 \times 2^4 \\ \hline & 1.0000000001100110011001100110011001100110011001100110_2 \times 2^4 \end{array}$$

De la misma manera se hace para restar 0.025 de 16.05:

$$\begin{array}{rl} & 1.0000000011001100110011001100110011001100110011001101_2 \times 2^4 \\ - & 0.0000000001100110011001100110011001100110011001100110_2 \times 2^4 \\ \hline & 1.0000000001100110011001100110011001100110011001100111_2 \times 2^4 \end{array}$$

Ya puede notarse que los resultados no son los mismos. Al hacer la resta final:

$$\begin{array}{rl} & 1.0000000001100110011001100110011001100110011001100111_2 \times 2^4 \\ - & 1.0000000001100110011001100110011001100110011001100110_2 \times 2^4 \\ \hline & 0.0000000000000000000000000000000000000000000000000001_2 \times 2^4 \end{array}$$

Este número al normalizarlo es:

$$1.0000000000000000000000000000000000000000000000000000_2 \times 2^{-48} \quad (4.13)$$

Es decir 2^{-48} que tiene un valor exacto decimal de:

$$3.552713678800500929355621337890625 \times 10^{-15} \quad (4.14)$$

El cual es el escurridizo valor del error mostrado en el cuadro 4.1.

4.2. Propuestas de Solución

Existieron varias propuestas al problema antes mencionado:

4.2.1. Propuesta de Solución 1. Aumento de Precisión

Si bien el sistema *Mathematica* maneja precisión arbitraria, ademas de la precisión por defecto, que es la provista por la máquina, existieron razones para sospechar que el aumento de precisión no necesariamente resolvería el problema. Si se observa el cuadro 4.3, en el cual se hace el mismo ejercicio en lenguaje *Java* que el anterior, pero utilizando tipos de datos de punto flotante de precisión sencilla, en lugar de precisión doble. En este ejercicio se observa que el error no se presenta. por lo que se llega a la sorprendente conclusión de que (para estos valores en particular), el error se presenta ¡al utilizar un formato de precisión superior!.

Esto no significa desde luego, que el formato de precisión arbitraria utilizado por el sistema *Mathematica* sea similar al IEEE 754 (en lo que a los errores detectados se refiere), sin embargo no se pudo encontrar en la documentación del sistema *Mathematica* la descripción técnica del formato utilizado para precisión arbitraria, por lo que no se tiene la seguridad absoluta de que funcione. De cualquier manera, sin importar el formato utilizado por *Mathematica* de representación de precisión arbitraria, al involucrarse valores *reales* cuyos valores en binario tienen representación fraccionaria de longitud infinita (periódica), es inevitable la pérdida de información aún aumentando la precisión arbitrariamente en el sistema de representación a utilizar.

```
float x = 16F + 0.025F;
float y = 16.05F - 0.025F;
System.out.println(y - x);

salida:
0
```

Figura 4.3: Ejercicio en *Java* Utilizando Aritmética de Precisión Sencilla.

4.2.2. Propuesta de Solución 2. Cambio de Unidades

Esto consiste en utilizar múltiplos de las unidades utilizadas a efecto de que todas las operaciones aritméticas se realizaran en aritmética entera. Por ejemplo, los datos utilizados, los que tiene mayor precisión, ésta es de 4 dígitos decimales, por lo que los datos a la entrada en los algoritmos pueden multiplicarse por $10,000$ y quedar siempre enteros. Esto equivale a que los algoritmos siempre utilizen "diezmilésimos" de los datos originales. A la salida, los resultados se dividen por $10,000$.

Esto sin embargo no funciona si en el algoritmo están presentes operaciones de división, que como se vió, no es una operación que presenta la propiedad de clausura (§3.3). De usarse aritmética entera, se produciría pérdida de información. En los algoritmos de cálculo de *diferencial* (§1.5.1), de cálculo de la tasa (§1.5.2), y cálculo de penalizaciones (§1.5.4) se encuentran operaciones de división, por lo que no es una solución viable.

Aunque no existieran operaciones de división en los algoritmos, el cambio de unidades afectaría significativamente la naturaleza elegante de "caja negra" prevista para los algoritmos. Habría que revisar ante un cambio de algoritmos si estos no involucran operaciones de división.

Otro hecho que afecta la naturaleza de caja negra, no en la parte elegante pero si funcionalmente (por lo que es mas severo) es si los algoritmos no involucran operaciones cuyo resultado sea afectado por el cambio de unidades. Por ejemplo, la operación *seno trigonométrico* no genera el mismo resultado si hay cambio de unidades (por ejemplo de grados sexagesimales a radianes). Esto implica que en estos casos sería necesario revisar los algoritmos para verificar si contienen operaciones de este tipo.

4.2.3. Propuesta de Solución 3. Uso de Aritmética Racional

Esta propuesta consiste en representar cada entidad en formato racional, es decir, en lugar de utilizar un valor en un formato que trata de representar de forma *aproximada* un valor con parte fraccionaria, se utilizan números racionales, los cuales nunca pierdan exactitud.

Esta solución resuelve la problemática presentada por las opciones anteriores. Sin embargo, no es mágica y tiene algunas limitantes. Se dedicará la sección siguiente para su discusión completa.

4.3. Aritmética Racional en el Sistema MexIBOR

Durante los cálculos de la tasa y penalizaciones, si bien existe una cantidad intensiva de operaciones numéricas, estas sólamente consisten en las cuatro operaciones básicas de adición, sustracción, multiplicación y división, y operaciones de comparación numérica.

4.3.1. Actualización del Sistema

Antes de la actualización, el sistema introducía valores numéricos a través de la conexión *MathLink* por medio de funciones del *API* cuyos argumentos son valores de tipo punto flotante, sin embargo, debido a (§3.3.2) no cualquier número real es representable, sino que tiene que ajustarse al mas cercano, y a (§3.4.2) que indica que un número de longitud finita en decimal, no necesariamente es de longitud finita en binario, se observa el ejemplo de la figura 4.4, en el cual un número decimal introducido como cadena (por ejemplo, capturado por un usuario), resulta tener pérdida de exactitud al ser convertido a valor de punto flotante. Si este número es introducido al *kernel* de *Mathematica* llegará ya con error.

A fín de introducir estos datos en forma racional sin recurrir a datos de tipo punto flotante, estos datos se convierten primero a tipo cadena de caracteres, y se introduce al kernel de *Mathematica* de la forma siguiente:

```
variable = Rationalize[ToExpression["valor como cadena"]]
```

La función `ToExpression[x]` obtiene la interpretación de la expresión dada como cadena de caracteres.

```
1   char *str = "0.2";
2   double x = atof(str);
3   printf(".50f", x);

salida:
0.20000000000000001110223024625156540423631668090820
```

Figura 4.4: Ejemplo de pérdida de precisión.

La función `Rationalize[x]` convierte la expresión x en forma racional exacta.[4]

Por ejemplo, el resultado de `Rationalize[ToExpression["0.5"]]` es el número racional 1/2.

De esta manera, las instrucciones contenidas en los *scripts* operan siempre con números racionales y no existe pérdida de información.

De forma inversa, los resultados estarán expresados en forma racional, por lo que al obtenerlos de regreso al sistema, hay que convertirse a números de punto flotante. Esto se logra recuperándolos por medio de:

```
N[x]
```

Que es la función de *Mathematica* para convertir a numérico cualquier resultado, por ejemplo `N[`$\frac{1}{2}$`]` resulta ser 0.5

Nótese que estos cambios se realizan en aquellas partes del código de interacción con el sistema *Mathematica*, para introducir o recuperar información, y que los *scripts* ¡no fueron en absoluto modificados! por lo que se conserva el principio de separación de algoritmos de los programas.

[4]Siempre y cuando el argumento `x` pueda así ser representado, por ejemplo, `Rationalize[`π`]` no puede ejecutarse satisfactoriamente.

Capítulo 5

Conclusiones

5.1. Sistema MexIBOR

La operación de la tasa MEXIBOR no es simple, sin embargo, como principio general de desarrollo de sistemas de información, el diseño del sistema debe ser tal que simplifique la comprensión del mismo, mucho antes de escribir la primer línea de código.

En el caso del sistema MexIBOR, la abstracción de la operación del sistema en estados y eventos resulta apropiado no sólo para la simplificación en la escritura de programas, sino también en la operación diaria del mismo a tal punto que el sistema ejecuta de forma autónoma y contínua por si solo. La única intervención humana es para realizar actualizaciones de parámetros o realización de pruebas.

La separación total de la parte matemática respecto de la programación permite la actualización de algoritmos de cálculo de la tasa sin modificar en absoluto los programas que conforman el sistema.

5.2. Principios de Utilización de Aritmética Racional

5.2.1. Uso de Memoria

Un aspecto importante en el uso de aritmética racional es que con cada operación aritmética, los números resultado tienden a ocupar mas memoria que la ocupada por lo operandos.

Suponga que tenemos dos números enteros a_1 y a_2. Ellos tienen un número

de cifras $cifras(a_1)$ y $cifras(a_2)$ respectivamente. Al realizar operaciones básicas de adición, sustracción y multiplicación entre ellos, el resultado contiene un número de cifras máximo que es como se indica en el cuadro 5.1.

operación	número máximo de cifras del resultado
$a_1 + a_2$	$max\{cifras(a_1),\ cifras(a_2)\} + 1$
$a_1 - a_2$	$max\{cifras(a_1),\ cifras(a_2)\}$
$a_1 \times a_2$	$cifras(a_1) + cifras(a_2)$

Cuadro 5.1: Número máximo de cifras requeridas despúes de realizar operaciones aritméticas con números enteros.

Suponga dos números racionales $\frac{a_1}{b_1}$ y $\frac{a_2}{b_2}$ y deseamos realizar sobre ellos operaciones aritméticas básicas de adición, sustracción, multiplicación y división y deseamos saber para cada operación el número de cifras (del numerador y del denominador) del resultado *en el peor de los casos*.

Para la adición/sustracción, el peor escenario es cuando los denominadores no tienen factores en común, en este caso:

$$\frac{a_1}{b_1} \pm \frac{a_2}{b_2} = \frac{a_1 b_2 \pm a_2 b_1}{b_1 b_2} \tag{5.1}$$

De este resultado, el peor escenario resulta nuevamente si el numerador y denominador no tienen factores en común, para que pudiera ser reducido a una forma mas simple.

Para la multiplicación/división, los resultados son:

$$\frac{a_1}{b_1}\frac{a_2}{b_2} = \frac{a_1 a_2}{b_1 b_2} \tag{5.2}$$

$$\frac{a_1}{b_1} \div \frac{a_2}{b_2} = \frac{a_1 b_2}{a_2 b_1} \tag{5.3}$$

Nuevamente, el peor caso es cuando el último resultado no puede ser reducido a una fracción mas simple.

5.2.2. Creación de Programas con Aritmética Racional Sin la Utilización de Sistemas Simbólicos

En el sistema MexIBOR, es el sistema *Mathematica* quien realiza los cálculos matemáticos, debido a la calidad de este sistema simbólico, y la modularización y elegancia que introduce su uso al proyecto. El sistema *Mathematica*,

operación	número máximo de cifras del numerador número máximo de cifras del denominador
$\frac{a_1}{b_1} + \frac{a_2}{b_2}$	$\begin{array}{c} max\{cifras(a_1) + cifras(b_2),\ cifras(a_2) + cifras(b_1)\} + 1 \\ cifras(b_1) + cifras(b_2) \end{array}$
$\frac{a_1}{b_1} - \frac{a_2}{b_2}$	$\begin{array}{c} max\{cifras(a_1) + cifras(b_2),\ cifras(a_2) + cifras(b_1)\} \\ cifras(b_1) + cifras(b_2) \end{array}$
$\frac{a_1}{b_1}\frac{a_2}{b_2}$	$\begin{array}{c} cifras(a_1) + cifras(a_2) \\ cifras(b_1) + cifras(b_2) \end{array}$
$\frac{a_1}{b_1} \div \frac{a_2}{b_2}$	$\begin{array}{c} cifras(a_1) + cifras(b_2) \\ cifras(a_2) + cifras(b_1) \end{array}$

Cuadro 5.2: Número de cifras en el numerador y denominador requeridas para realizar operaciones aritméticas entre números racionales.

al igual que otros sistemas simbólicos, no es barato, pero para la envergadura del sistema MexIBOR en tamaño y operación crítica, resulta ser una inversión justificada.

Sin embargo, para proyectos que requieren exactitud aritmética, pero cuyas circunstancias no jsutifican la compra de un sistema simbólico, es posible utilizar aritmética racional intrínseca a los programas mismos, A continuación se muestran las características y condiciones mínimas a satisfacerse para ello. No se muestran programas fuente (nunca han sido desarrollados por el autor), pero sí los lineamientos y consejos para iniciar dicha implementación.

La forma mas sencilla es no utilizar los tipos de datos de punto flotantes provistos por el lenguaje de programación, sino crear estructuras de datos propias. En lenguajes programados a objetos es recomendable crear clases, en aquellos que no los son se puede utilizar registros, estructuras o el tipo de datos de agrupación de datos primitivos.

Estas clases, registros o estructuras simplemente contendrían dos valores de tipo entero, el de mayor amplitud soportada por el lenguaje (preferentemente de longitud arbitraria, por ejemplo la clase `BigInteger` del lenguaje *Java*). Uno de ellos representando al numerador y otro representando al denominador del número racional.

Las operaciones aritméticas básicas de suma, resta, multiplicación y división deben ser implementadas, ya sea como funciones miembro de la clase citada (para lenguajes orientados a objetos), o funciones autónomas para los que no lo son. Los lenguajes que soportan sobrecarga de operadores (como C++) pueden aprovechar esta característica para que los programas fuente

tengan mayor claridad.

Resulta conveniente que en cada operación aritmética realizada, el número se reduzca a una fracción mas simple (en caso de que se pueda). Cualquier expresión racional $\frac{a}{b}$ queda reducida a su forma mas simple por medio de:

$$\frac{a/MCD(a,b)}{b/MCD(a,b)} \tag{5.4}$$

Por razones de eficiencia, no conviene realizar la operación exactamente como en la expresión 5.4, sino realizarlo en los siguientes pasos:

1. Determinar el máximo común divisor (MCD) del numerador y el denominador $MCD(a,b)$. Si este tiene un valor de 1, entonces el número ya se encuentra en su forma mas simple. El numerador y el denominador son numeros primos entre sí, o equivalentemente, no tienen factores comúnes (ademas del factor obvio 1).

2. En caso de que el máximo común divisor tenga un valor diferente a 1, dividir tanto al numerador como al denominador por este número.

La determinación misma del máximo común denominador puede parecer una operación no muy eficiente, sin embargo podemos hacer uso del algoritmo de Euclides, mostrado en la figura 5.1, basado en el hecho siguiente:

Hecho Si a y b son enteros positivos y $a > b$ entonces
MCD(a, b) = MCD(b, a mod b)

ENTRADA:	dos enteros no negativos a y b, siéndo $a \geq b$
SALIDA:	el máximo común divisor de a y b
1. mientras $b \neq 0$ realizar:	
	$r \leftarrow a$ mod b
	$a \leftarrow b$
	$b \leftarrow r$
2. regresar (a)	

Figura 5.1: Algoritmo de Euclides para determinar MCD de dos números.

El algoritmo de Euclides realiza un número máximo de $O(\lg^2 n)$ operaciones aritméticas binarias.

Esta forma de manipulación numérica presenta varias ventajas respecto a la representación típica. Una gran ventaja que tiene esta manipulación numérica es que *todas* las operaciones aritméticas básicas de adición, sustracción, multiplicación y división *sí* presentan la propiedad de clausura.

La implementación debería incluír mecanismos para convertir de un número de punto flotante o cadena (que represente a un número real) al tipo racional, y viceversa (de racional a punto flotante o cadena).

Aún un número real con parte fraccional infinito periódico puede transformarse a forma racional *sin pérdida de información* de la siguiente manera (se presuponen números sin parte entera, y la parte fraccionaria periódica desde el primer decimal, pero el método mostrado aquí puede extenderse fácilmente):

$$X = 0.\overline{a_1 a_2 \dots a_n} = 0.(a_1 a_2 \dots a_n)(a_1 a_2 \dots a_n) \dots \tag{5.5}$$

Es decir, el número consta de n cifras que se repiten indefinidamente. Entonces:

$$10^n X = (a_1 a_2 \dots a_n).\overline{(a_1 a_2 \dots a_n)} \tag{5.6}$$

Sustrayendo el número original X:

$$10^n X - X = (10^n - 1)X = a_1 a_2 \dots a_n \tag{5.7}$$

Y por lo tanto:

$$X = \frac{a_1 a_2 \dots a_n}{10^n - 1} \tag{5.8}$$

Como ejemplo, para representar en forma racional al número $0.123123\ldots = 0.\overline{123}$ procedemos como sigue:

$$0.\overline{123} = \frac{123}{999} = \frac{41}{333} \tag{5.9}$$

Cambiar un programa existente sería un problema de cambio de sintaxis, mas que un de cambio de semántica o lógica.

5.2.3. Cuándo *NO* Utilizar Aritmética Racional

Es importante hacer notar algunas condiciones bajo las cuales el uso de aritmética racional no funciona:

Cuando la exactitud no es requerida En este caso la aritmética racional sólo elevaría la complejidad del sistema informático.

Si intervienen resultados no representables como números racionales Esto aplica si se desea construír programas con soporte de aritmética racional (§5.2.2), ya que para conservar la exactitud, sería necesaria realizar manipulación simbólica. Evidentemente, si se tiene un manejador simbólico (como *Mathematica*) no representa un problema (siempre y cuando se observen los requisitos mostrados en el apartado §(8)).

Cuando interviene una cantidad excesiva de cálculos Por lo visto en el apartado §(5.2.1), en cada cálculo, intervienen generalmente números enteros de mayor magnitud. Si el número de cálculos es excesivo puede llegarse al límite de representación, aún en sistemas con estructuras de datos enteras de longitud arbitraria.[1]

[1] Ya que *arbitrario* en este caso, no significa *infinito*, sino no limitado por razones de diseño. De existir una limitación será la impuesta por otros recursos, por ejemplo, memoria del equipo.

Bibliografía

[ABM01] Asociación de Banqueros de México A.C.: *Tasa MexIBOR*, http://www.abm.mx/Mexibor/Principal.html, 2001

[Bei64] Beiler, Albert H: *Recreations in the Theory of Numbers*, 2^{nd} edition, Dover Publications, Inc. 1964

[Gar86] Gardner, Martin: *Miscelánea Matemática*, Biblioteca Científica Salvat, 1986

[Kru98] Kruglinski, David: *Inside Visual C++*, Microsoft Press, 1998

[Nor44] Northrop, Eugene P.: *Riddles in Mathematics, A Book of Paradoxes*, D. Van Nostrand Company, Inc., 1944

[Pet97] Petzold, Charles: *Programming Windows 95*, Mc Graw Hill, (1997)

[REU01] REUTERS: *Contributions Library*, Release 1.02

[REU02] REUTERS: *REUTERS Dynamic Data Exchange*, Version 3.5

[REU03] REUTERS: *Triarch Programming Guide*, *Reuters SSL Developer's Guide*, Release 4.0, (2001)

[REU04] REUTERS: *Triarch Programming Guide*, *Reuters SSL Reference Manual*, Release 4.0, (2001)

[Wil94] Wilf, Herbert S: *Algorithms and Complexity*, Internet Edition, University of Pennsylvania, (1994)

[Wol99] Wolfram, Stephen: *The Mathematica Book*, Fourth edition, Cambridge University Press, (1999)

[Van87] Van Horne, James C.: *Administración Financiera*, décima edición, Prentice Hall, (1987)

Índice alfabético

Créditos

Fórmulas matemáticas y algoritmos de cálculo: Mitch Stonehocker.
Responsable del proyecto (*Chase Manhattan Bank*): Gabriel Vales.
Responsable del proyecto (*REUTERS*): Marsella Camero.
Desarrollo de programas: Laurence Ruiz Ugalde.
Desarrollo y documentación de scripts: Laurence Ruiz Ugalde.
Apoyo especial: Fabiola Hernández, Berenice Bojórquez, Stefan Feczko, José Luiz Valdéz, Rocío Muñoz; *REUTERS*.