



UNIVERSIDAD DE MURCIA

FACULTAD DE INFORMÁTICA

OBANSoft: aplicación para el análisis bayesiano objetivo y subjetivo. Estudio de su optimización y paralelización

Tesis de Máster

Máster en Informática y Matemáticas Aplicadas a
Ciencias e Ingeniería

Autor:

Manuel Quesada Martínez

Directores:

Dr. Domingo Giménez Cánovas y Dra. Asunción Martínez Mayoral

Murcia, 5 de julio de 2010

Agradecimientos

En primer lugar quisiera agradecer a Domingo Giménez y Asunción Martínez la oportunidad que me han brindado para comenzar este proyecto y aprender junto a ellos durante los próximos años.

A la Universidad de Murcia, Facultad de Informática y a mi ña-carne Fran Moreno, por haberme dado la oportunidad de aprender con ellos Informática.

A mis padres, por educarme, aconsejarme y apoyarme siempre, transmitiendo la tranquilidad de saber que puedes contar con ellos.

A mi abuela Mercedes, por regalarme la oportunidad de tener una segunda madre. Y a mi abuelo Juan Manuel porque pese a la distancia, él siempre logra ayudar y agradar con sus consejos de curtido veterano.

Al resto de mi familia, en especial a mi hermana y prima pequeña Merce porque me han enseñado a disfrutar compartiendo todo lo que aprendemos.

A todos mis profesores, desde el colegio hasta la universidad, y en especial a mi señorita Chon, por enseñarme lo más difícil: leer, escribir y sumar.

A todos mis amigos de siempre, Patri, Javi, Juanito... y a todos los demás que siempre estáis ahí.

Agradecer especialmente a María porque, a pesar de aparecer más tarde que el resto, tu apoyo incondicional ha sido y será fundamental durante cada día de mi vida.

Gracias.

Índice

1. Introducción	7
1.1. Trabajos relacionados	8
1.2. Metodología	9
1.2.1. Diseño del software <i>OBANSoft</i>	9
1.2.2. Paralelización del código	10
1.3. Herramientas	11
1.4. Objetivos	12
1.4.1. A corto plazo	12
1.4.2. A largo plazo	13
1.5. Contenido del documento	13
2. Estado del arte	14
2.1. Alternativas y lenguajes para el análisis bayesiano	14
2.2. Interfaces gráficas para <i>R</i>	15
2.2.1. <i>Rcomander</i> (<i>A basic statistics GUI for R</i>)	15
2.2.2. <i>JGR</i> (<i>Java GUI for R</i>)	16
3. Diseño de la aplicación <i>OBANSoft</i>	17
3.1. Requisitos, funcionalidad y módulos	17
3.1.1. Creación y edición de bancos de datos (<i>DataFrame</i>)	18
3.1.2. Transformaciones sobre los <i>DataFrame</i>	19
3.1.3. Simulación de datos	19
3.1.4. Descriptivas numéricas y gráficas	19
3.1.5. Inferencia bayesiana y modelos	20
3.2. Aspectos más relevantes del diseño	20
3.2.1. Aplicación del modelo Modelo-Vista-Controlador (<i>MVC</i>)	20
3.2.2. Principales objetos e interacción entre ellos	22
3.2.3. Algoritmos bayesianos. Integración de tecnologías	24
3.2.4. El R-Modelo y su integración con <i>R</i>	25
3.2.5. El log de la aplicación y otras cuestiones	26
3.3. Recursos disponibles y futura extensión	27

4. Optimización y paralelización de los algoritmos	27
4.1. Estudio del rendimiento de los algoritmos de simulación	27
4.2. Introducción del paralelismo	30
4.3. Estudio experimental del paralelismo	32
5. Conclusiones y trabajos futuros	33
Bibliografía	35
A. Anexo I: apéndice de funciones bayesianas implementadas	37
B. Anexo II: póster <i>OBANSoft</i> en el <i>ISBA World Meeting 2010</i>	37

1 Introducción

Es un hecho constatable la cada vez más patente implantación de las técnicas de análisis estadístico bayesiano en todos los campos del conocimiento. Estos modelos tienen aplicaciones directas en multitud de campos científicos (como la Epidemiología, Medio Ambiente, Investigación Clínica, etc.) y problemas (selección de variables, predicción en problemas espaciales, redes neuronales, machine learning, etc.). También es una necesidad, cada día más notable, la elaboración de un *software* bayesiano capaz de resolver, de forma automatizada y desde un punto de vista realmente objetivo, sin la intervención de información previa subjetiva por parte del modelador, los diversos problemas estadísticos que pueden surgir en la práctica.

Entre los objetivos de esta tesis de máster, se encuentra el de llenar este vacío e implementar progresivamente modelos y análisis bayesianos objetivos (además de los subjetivos) habituales, integrándolos en una aplicación de escritorio operativa para comenzar a asimilar módulos de análisis bayesianos y dar respuesta a las necesidades existentes. La aplicación será el punto de partida para su extensión con los trabajos de investigación derivados de esta tesis de máster (por ejemplo, el desarrollo de modelos o algoritmos optimizados y paralelizados). Aprovechando las iniciales de “*Software for Objective Bayesian Analysis*” nombraremos, a partir de este momento, a la aplicación como *OBANSofT*.

Entre la funcionalidad implementada en la aplicación se encuentran los algoritmos de simulación para distintos tipos de distribuciones (continuas y discretas). El algoritmo de algunos de estos simuladores será el resultado de componer dos o más simulaciones sencillas. Estas funciones compuestas presentan en su funcionamiento características similares con los algoritmos de cadenas de Markov Monte Carlo (*MCMC*), que pretenden ser la línea de investigación que dará continuidad a esta tesis de máster. Los algoritmos *MCMC* presentan problemas computacionales de alto coste, y por tanto será interesante optimizarlos e introducir técnicas de programación paralela en su implementación. El resultado de estos algoritmos paralelos será conseguir beneficios tanto en el tiempo de ejecución como en la calidad de las simulaciones.

A medida que avancemos en el desarrollo de la tesis de máster, se programarán e incorporarán en la aplicación los primeros algoritmos de simulación. Para comparar el funcionamiento entre todos ellos, se utilizará una metodología de trabajo que permita experimentar y descubrir cuales de ellos consumen más recursos y por tanto merece más la pena paralelizarlos.

La capacidad de extensión del programa desarrollado obliga a realizar su diseño de forma modular, definiendo claramente su funcionalidad y los módulos que lo integran. La aplicación y librerías desarrolladas serán de libre distribución y código abierto. Por este motivo, será deseable dotarlo de interfaces bien definidas que faciliten el acoplamiento con los nuevos paquetes y módulos.

La optimización y paralelización de los algoritmos podría realizarse en diferentes arquitecturas y con diferentes lenguajes de programación, en función de las características de la distribución simulada. Esta mezcla de entornos justificará el estudio de cómo enlazar el programa con otras rutinas implementadas para diferentes plataformas y sistemas;

es decir, la aplicación debería ser capaz de invocar en una capa inferior a rutinas programadas en *C*, *C++*, *Fortran*, *R*... incluso cuando utilizan librerías para optimizar los cálculos, por ejemplo librerías paralelas como *OpenMP*, *MPI* o *CUDA*. Además, dado que los usuarios finales de la aplicación serán expertos de diversas disciplinas y no necesariamente en paralelismo, podríamos tratar de incluir técnicas de auto-optimización de código que liberen a estos usuarios de la configuración de parámetros asociados a estas nuevas rutinas paralelas.

1.1 Trabajos relacionados

Este trabajo se realizará en el Grupo de Computación Científica y Programación Paralela de la Universidad de Murcia (GCCPPUM), que tiene experiencia en el desarrollo y optimización de código paralelo, incluyendo técnicas de autooptimización, y en la aplicación de la computación paralela en diversos campos científicos [1].

También se colaborará con el Grupo de Estadística Bayesiana de la Universidad Miguel Hernández (GEBUMH) que tiene experiencia en el desarrollo de códigos de simulación aplicables a la resolución de análisis bayesianos en problemas agronómicos, clínicos y medioambientales [2, 3, 4].

Tras una primera revisión de la literatura, nos encontramos ante la perspectiva, cada vez más patente, de la implantación extensiva de las técnicas de análisis estadístico bayesiano en la práctica estadística para todos los campos de conocimiento experimental. Si hay algo que ha permitido la extensión en la aplicación de análisis estadísticos bayesianos en diversos ámbitos [5, 6] ha sido, sin lugar a dudas, el avance computacional producido desde los años 80. Las capacidades del hardware han permitido a los investigadores proponer y testar nuevas técnicas y algoritmos de simulación, para conseguir soluciones a los problemas no analíticos que surgen habitualmente en estos análisis bayesianos.

En 1989 se inicia el proyecto *BUGS*, que posteriormente da lugar a *Winbugs* [7]. Con el tiempo, este proyecto se ha convertido en un software bayesiano que ha contribuido notablemente a la extensión de los análisis bayesianos en la práctica cotidiana de muchos ámbitos. La principal carencia de esta aplicación es que, al venir implementado sobre modelos subjetivos, no permite una modelización genuinamente objetiva.

La relevancia de la estadística bayesiana objetiva radica en las posibilidades que da al investigador para no comprometer las inferencias con la modelización subjetiva de la información previa, utilizando una modelización a priori que proviene de una regla formal matemática [8, 9], y que permite concluir, a diferencia de las técnicas frecuentistas, en términos de probabilidad y plantear y resolver modelizaciones complejas. La referencia [10] contiene el catálogo más extenso hasta la fecha, de las diferentes distribuciones previas no informativas que permiten derivar verdaderos análisis bayesianos objetivos en diferentes modelizaciones, y constituye el manual base de referencia para desarrollar este trabajo.

1.2 Metodología

Para elaborar esta tesis de máster deberemos abordar diversas áreas de conocimiento. Este hecho ha llevado a dividir la metodología en cuatro puntos principales, y para cada uno se utiliza una metodología particular:

- **Parte I:** tomando como punto de partida [10] se ha elaborado un catálogo con las operaciones bayesianas soportadas por la aplicación: transformaciones, simuladores, modelos e inferencia (Anexo I).
- **Parte II:** partiendo de las necesidades extraídas en el documento del punto anterior, se decidió qué tecnologías y recursos utilizar para diseñar e implementar la aplicación y su librería de funciones.
- **Parte III:** se diseñó e implementó la librería estadística y su interfaz de usuario. La combinación de los dos objetos perseguirá simplificar (respecto a las herramientas disponibles) y guiar el proceso de inferencia utilizando los modelos bayesianos detectados en el primer punto.
- **Parte IV:** se ha estudiado el rendimiento de la aplicación y mejorado en las partes más adecuadas.

La primera parte se ha realizado conjuntamente con los expertos estadísticos, que son los que tienen conocimientos del área para decidir que tipo de análisis son los más utilizados e interesantes de incluir en la aplicación. La segunda parte ha sido un estudio del estado del arte y las diferentes alternativas disponibles para utilizar o implementar las operaciones del catálogo construido en el primer punto. Para las dos últimas partes merece especial interés comentar con más detalle la metodología utilizada que a continuación se explica.

1.2.1 Diseño del software *OBANSoft*

Para un usuario nada acostumbrado a trabajar con estadística bayesiana, entenderla supone un reto difícil por la cantidad de documentación y conceptos a asimilar. Para facilitar la realimentación entre los expertos estadísticos y los informáticos, vamos a utilizar un método ágil para el diseño del software. Descartamos utilizar un método pesado por no tratarse de un proyecto que necesite: un sistema crítico, involucre grandes equipos de desarrollo o sea necesario desarrollar una amplia y minuciosa documentación. Utilizar un método ágil apoyado en el uso de un prototipo funcional y evolutivo, facilitará la extracción de requisitos a medida que avanzamos en su desarrollo. La mayoría de estos requisitos son difíciles de detectar sin tener más referencias que las ideas iniciales.

Vamos a utilizar una metodología basada en eXtreme Programming (*XP*) [11]. El desconocimiento por parte de los expertos involucrados, bien del campo de la estadística o bien del campo de la informática, va a condicionar que exista una comunicación fluida

entre ellos. Sin embargo, la experiencia de los investigadores estadísticos en la programación de algoritmos en *R* facilitará la revisión del nuevo código programado. El desglose de la metodología seguida es:

- En unas primeras sesiones se analizaron los principales apartados en los que se quería organizar la aplicación. Analizamos un prototipo inacabado de una aplicación de la que dispone el GEBUMH. Como resultado de este análisis, se pretende adquirir una primera idea de la manera de trabajar e inferir con bancos de datos. Los apartados en los que se decidió organizar el programa serán denominados a partir de este punto como los módulos de la aplicación.
- Una vez extraídos y enumerados los diferentes módulos generales que constituyen la aplicación, se ha repetido el siguiente proceso para cada uno de ellos:
 - Extracción de requisitos para dicho módulo y diseño de los elementos software que permitirán modelar este comportamiento.
 - Análisis de las posibilidades que ofrece *R* para realizar dichas operaciones. En caso de que alguna funcionalidad haya sido recogida en el Anexo I, actualizaremos el documento con el algoritmo de *R* que lo implementa.
 - Actualización de una librería *Java* que permita realizar las operaciones estadísticas. En este caso se implementarán los algoritmos desde cero o se invocará a *R* para su ejecución (si se detectaron funciones en el punto anterior).
 - Diseño y elaboración de formularios que permitan implementar la interfaz gráfica que utilice las funciones de la librería del punto anterior.
 - Realización de pruebas unitarias para la funcionalidad implementada en los dos puntos anteriores, así como otras pruebas de integración con los módulos implementados en iteraciones anteriores.
- Una vez finalizados todos los módulos se ha revisado la aplicación completa. Se ha ofrecido a distintos expertos en el área para su prueba y propuesta de modificaciones.
- Se han implementado las modificaciones adecuadas sugeridas por los investigadores.

A través de esta metodología se ha conseguido una primera versión de la aplicación que permite realizar los primeros análisis bayesianos objetivos y subjetivos sin necesidad de integrar más herramientas que *OBANS*Soft. Opcionalmente también se podrá utilizar la librería *Java* (sin el entorno gráfico) para ser importada por otros proyectos que necesiten la ejecución de los algoritmos desarrollados.

1.2.2 Paralelización del código

Pese a que la aplicación está formada por varios módulos con diferentes objetivos, vamos a centrar el estudio de la paralelización en aquellos módulos que realmente ejecutan algoritmos matemáticos que podrían suponer problemas en los tiempos de ejecución. Por ejemplo, para un módulo de gestión y manipulación de datos no se estudiará su

optimización por no aportar ningún avance en la línea de investigación que derivará de esta tesis de máster (paralelización de algoritmos *MCMC*). Concretamente, se ha centrado el estudio en los módulos que involucren a los algoritmos de simulación o modelos de inferencia bayesiana.

Para la implementación de los algoritmos de simulación existirán dos posibilidades: implementar los algoritmos en *R* apoyándonos en sus rutinas y librerías, o utilizar rutinas desarrolladas en otros lenguajes (*C*, *C++*, *Fortran* básicamente), para conseguirlos. Para cualquiera de los casos, se dispondrá del código fuente para poder estudiar su paralelización.

Con el fin de proceder de modo racional, en una primera fase, la abordada en esta tesis de máster, se optó por utilizar exclusivamente las funciones existentes en *R* y dejar, para un trabajo posterior, la búsqueda y trabajo con código fuente y código procedente de otros lenguajes. Se enmarca como fin global, el de estudiar la optimización de los algoritmos utilizados, provengan de donde provengan, y asumir pues, una caja negra sobre la que se va a desarrollar el trabajo, independientemente de encontrar o no el código fuente.

La metodología seguida es:

- Se ha realizado un análisis experimental con todos los simuladores soportados en la aplicación para medir sus tiempos de ejecución y recursos consumidos.
- Se han evaluado los resultados obtenidos de los experimentos anteriores y seleccionado las funciones que más tiempo de ejecución consumen para paralelizarlas.
- Para cada una de las funciones a paralelizar, se ha estudiado su algoritmo y elaborado una versión paralela en pseudocódigo.
- Se adaptó la versión paralela para ser implementada en un entorno de memoria compartida.
 - Si el algoritmo de simulación está implementado en *R* utilizaremos directamente alguna librería que permita ejecutar código *R* en paralelo para memoria compartida.
 - Si el algoritmo de simulación está implementado desde cero en *C* se paralelizará utilizando el estándar *OpenMP*.
- Hemos realizado el análisis experimental en una máquina con un procesador multinúcleo. Se ha probado la ejecución de los algoritmos paralelizados oscilando el número de procesadores en el intervalo [1, numProcesadores] incrementándolos de uno en uno.

1.3 Herramientas

Se han utilizado las siguientes herramientas para desarrollar la aplicación *OBANSofT*. En una primera fase se ha elaborado la aplicación gráfica de usuario e implementado los algoritmos estadísticos a utilizar. Para ello haremos uso de las siguiente herramientas:

- Para el desarrollo del proyecto *Java + GUI* vamos a utilizar el entorno de trabajo **NetBeans**. Concretamente utilizaremos el paquete de descarga “*Java SE*” que incluye las tecnologías “*NetBeans Platform SDK*” y “*Java SE*”. La versión utilizada será la 6.8, y lo implementaremos paralelamente en un sistema “*Windows XP*”, “*Ubuntu 8.10*” y “*MacOS Leopard*”.
- Para la elaboración de la interfaz gráfica vamos a utilizar las librerías de componentes gráficos y controles “**Java Swing**”. Nos apoyaremos a su vez en el *framework* disponible en la herramienta del punto anterior para manipular los formularios.
- Utilizaremos el paquete estadístico **R** para aprovechar los algoritmos que ya han sido programados y puedan ser reutilizados. La versión de *R* utilizada ha sido la 2.10.0, y en sus versiones para los tres sistemas operativos mencionados en el primer punto de esta lista. Algunas operaciones de *R* utilizadas, no se encontraban en el paquete base; y para utilizarlas hemos cargado las librerías: “*car*”, “*adapt*”, “*gplots*”, “*ggm*”, “*actuar*”, “*degreenet*”, “*dataframes2xls*”, “*xlsReadWrite*”, “*foreign*”.
- Para enlazar *Java* con *R* vamos a utilizar la librería **JRI** (*Java/R Interface*).

En una segunda fase, se ha realizado un estudio experimental del rendimiento de los algoritmos de simulación utilizados en la aplicación. Para ello utilizaremos la máquina multicore y las librerías siguientes:

- Para paralelizar los algoritmos de simulación, utilizaremos la librería **SnowFall** [12]. Esta librería permite definir “*wrappers*” para programar las secciones de código que se desean paralelizar.
- La evaluación y estudio de los algoritmos de simulación (incluida su paralelización) se ha llevado a cabo en la máquina del grupo *GCCPPUM: Luna.inf.um.es*. Esta máquina multicore está compuesta por 4 núcleos pertenecientes a un procesador Intel Quad Core, con 4.0 GB de RAM.

1.4 Objetivos

Vamos a diferenciar entre los objetivos a corto y largo plazo:

1.4.1 A corto plazo

Los objetivos a corto plazo han sido alcanzados al finalizar este trabajo de introducción a la investigación. Los enumeramos como:

- *Obj1*: formación en el campo de la estadística bayesiana. Familiarización con el tratamiento de bancos de datos, descriptivos y modelos de inferencia.

- *Obj2*: elaboración de un documento de referencia con la funcionalidad que va a soportar la aplicación. El manual incluirá para cada una de las funciones si se puede realizar con *R*, y en caso afirmativo qué sentencia y paquete utilizar (Anexo I).
- *Obj3*: elaboración de una librería *Java* que implemente las operaciones recogidas en el *Obj2*. Las funciones *Java* invocarán a las rutinas *R* en caso de existir, y en caso contrario serán programadas directamente en *C*.
- *Obj4*: diseño de los formularios y de la aplicación gráfica de escritorio que soporte la funcionalidad recogida en el *Obj2* e implementada en el *Obj3*.
- *Obj5*: paralelización de tres rutinas que supongan el núcleo computacional de la aplicación.
- *Obj6*: estudio de la paralelización realizada en al menos un sistema con arquitectura de memoria compartida.

1.4.2 A largo plazo

Por otro lado, identificaremos como objetivos a largo plazo nuevas líneas de investigación que puedan surgir a partir de este trabajo. Las enumeramos en orden cronológico de lo que sería más inmediato incluir a lo que menos.

- Extender la paralelización y optimización a todos los simuladores que contiene esta primera versión de la aplicación. Estudiar otros puntos de la aplicación sensibles de paralelizar y optimizar.
- Añadir nuevos modelos de inferencia bayesiana. Sobre todo interesaría proporcionar una aplicación completa que facilite hacer análisis y las simulaciones de cadenas de *Markov* (*MCMC*) a la comunidad de estadística bayesiana. Estos algoritmos suponen el paso siguiente a los modelos implementados en esta tesis de máster.
- Incluir opciones de autooptimización en la instalación de *OBANSofit*. Estudiar la paralelización en otras plataformas y con otros paradigmas de programación paralela: *MPI*, *CUDA*, computación heterogénea...
- Estudiar la portabilidad de la aplicación de escritorio a una aplicación web, que permita la explotación de la herramienta por “*Cloud Computing*”.

1.5 Contenido del documento

En las siguientes secciones de este documento se recoge un resumen del trabajo que hemos realizado a lo largo de la tesis de máster. En la segunda sección se recoge un análisis del estado del arte y las alternativas que actualmente existen para realizar análisis bayesianos. En la tercera sección se recogerán los aspectos relativos al diseño e implementación

de la aplicación *OBANS*soft, tanto de la librería como de la aplicación de escritorio; recogerá aspectos desde su diseño, hasta su implementación y futura extensión. En la cuarta sección se comentará como se ha realizado el estudio y paralelización de los algoritmos de simulación. Por último se enuncian las conclusiones y trabajos futuros surgidos a partir de esta tesis de máster.

2 Estado del arte

En este apartado vamos a analizar que posibilidades existen para realizar, los análisis de datos bayesianos objetivos, y prestaremos especial interés a las posibilidades de realizarlos apoyados por una herramienta con interfaz gráfica de usuario.

2.1 Alternativas y lenguajes para el análisis bayesiano

A pesar de existir en el mercado un número considerable de paquetes estadísticos, los requerimientos específicos de los análisis bayesianos sólo son viables, a día de hoy, a través de ciertas librerías de *SAS* [13], de *R* [14] y mediante el software bayesiano *WinBugs*.

Entre los dos programas estadísticos genéricos, *SAS* y *R*, este último se beneficia del prolífico uso que se le ha venido dando desde finales de los años 90 además de ser un programa muy estable desarrollado en *C++*. *Winbugs*, por otro lado, es un software diseñado en exclusiva para resolver análisis bayesianos desde una perspectiva subjetiva en la que el modelizador siempre dispone de información previa.

A la hora de abordar la generación de un software bayesiano que permitiera llevar a cabo la realización de análisis bayesianos objetivos, esto es, análisis que parten de una carencia absoluta o casi absoluta de información previa, *R* resultaba el mejor candidato. Sus características, en las que profundizamos a continuación, su extensión entre la comunidad científica y experimental, y el hecho de que ya se hayan venido implementando y asimilando librerías genuinamente bayesianas, han sido factores decisivos para justificar su utilización como motor de cálculo en nuestra aplicación. El proyecto *R* es actualmente el paquete estadístico libre más utilizado en todos los ámbitos de investigación estadística. Es una herramienta multiplataforma que puede ser ejecutada en los principales sistemas operativos (*Windows*, *Linux* y *MacOS*). Existen una multitud de paquetes programados por usuarios e investigadores que engloban un amplio abanico de operaciones estadísticas. Las funciones ofrecidas por estos paquetes abarcan desde la gestión de bancos de datos, descriptivas numéricas y gráficas (univariante y bivariantes) hasta librerías bayesianas que permiten realizar análisis más complejos (objetivos o subjetivos).

R es en sí un lenguaje de programación de alto nivel, basado en *C++*, dotado de características y funciones que facilitan el trabajo con los datos; y que además no requiere del usuario excesivos conocimientos en programación. Otro punto a su favor es la posibilidad de enlazarlo con rutinas programadas en *C* o *Fortran*. La mayoría de los paquetes

que forman la base de R están desarrollados en estos dos lenguajes. Cada paquete define su propia interfaz utilizando sintaxis del lenguaje R ; e internamente se mapean a ficheros fuentes programados en C (u otros lenguajes). Los paquetes tienen licencia GNU y por tanto tendremos disponible en todo momento su código fuente. Esto será útil cuando interese estudiar los algoritmos para optimizarlos y proceder a su paralelización.

Respecto a los entornos gráficos de las aplicaciones comentadas, todas las soluciones de pago implementan una interfaz gráfica que simplifica el proceso de análisis para los usuarios menos acostumbrados a trabajar con sentencias de programación. El proyecto R , por defecto, basa todo su funcionamiento en una consola de comandos. No existe ninguna aplicación gráfica nativa que se distribuya con el paquete base; aunque como resultado de su gran aceptación entre la comunidad científica, han surgido numerosas extensiones para instalar como módulos de esta base que cumplen este cometido. Por disponer de diferentes posibles extensiones de $GUIs$, se va a realizar un estudio entre ellas para estudiar sus diferencias.

2.2 Interfaces gráficas para R

Tras analizar los últimos artículos (revistas y congresos) y otra bibliografía sobre aplicaciones estadísticas, se concluye que la mayoría de usuarios de R que quieren utilizar una interfaz gráfica de usuario utilizan uno de estos dos paquetes: ***RComander*** [15] (en su última versión 1.5-4) o ***JGR*** [16] (en su última versión 1.7-0).

2.2.1 *Rcomander (A basic statistics GUI for R)*

El paquete *Rcomander* es el más extendido cuando se quiere trabajar con funciones de R de manera visual. Esta herramienta surgió en el año 2005. Podemos encontrar en su primer artículo [17] el marco científico en el que comenzó. La motivación de su creador fue similar a la de este proyecto: R pese a ser una buena herramienta con miles de instrucciones, funciones, paquetes..., carecía de una interfaz gráfica para simplificar el modelado estadístico a los usuarios poco, o nada, interesados en aprender un lenguaje de programación.

Para el **proceso de instalación y uso**, su autor *John Fox* la desarrolló para distribuirla como un paquete más de R ; simplemente instalándola (desde los propios servidores del proyecto R) e invocándola desde la terminal de comandos ya se puede utilizar con la sentencia `library("Rcmdr")`.

Respecto a la **tecnología** utilizada para su desarrollo, la librería *JGR* está construida basándose en el paquete *TCLTK* (facilitando una interface al toolkit *Tcl/Tk GUI*). *Rcmdr* es una interfaz que permite realizar operaciones utilizadas sobre todo con estadística clásica (frecuentista), pero no se ha dedicado esfuerzo para facilitar la aplicación de análisis bayesianos.

Respecto a su **capacidad de extensión**, el autor lo dotó de interfaces para extenderlo con nuevos menús y funcionalidad. Sus menús están definidos en un fichero de texto

plano (“*Rcmdr-menus.txt*”) que reside en el directorio de configuración del paquete. Modificando este fichero, los cambios serían trasladados al menú principal de la aplicación. El formato de este fichero se encuentra definido en su documentación. Se han realizado algunas pruebas para estudiar posteriormente su viabilidad como paquete de partida de *OBANSOft*.

En líneas generales *Rcmdr* es una interfaz robusta y extendida para dotar a *R* de una *GUI* con operaciones de gestión de datos, descriptivas y modelado clásico. Sin embargo, esta interfaz no acaba de ofrecer al usuario un entorno especialmente amigable y en cualquier caso, la posible integración de análisis bayesianos habría de ejecutarse a través de la implementación de módulos/librerías específicas que habrían de asumir las bases de las previamente desarrolladas, limitando así posibles pretensiones de accesibilidad o presentación de resultados.

2.2.2 *JGR (Java GUI for R)*

JGR es otra opción para extender el *R* base con una interfaz de usuario más rica y descriptiva. Su primera versión data del año 2004, y se presentó como herramienta en su artículo de presentación [18]. El **proceso de instalación y uso** es idéntico al que ofrece *RComander*, se comporta completamente integrado con el *kernel* de *R* sobre el que se ha instalado. A diferencia del paquete del apartado anterior, *JGR* se basa en **tecnología Java**. Concretamente, cimenta su integración con *R* en otros dos paquetes que actúan una capa por debajo: “*rJava*” y “*JRI*”. Para gestionar las ventanas y los controles de la *GUI*, el paquete *rJava* utiliza los *frameworks* “*Java Swing*” y “*AWT*”. La labor de los paquetes de esta capa inferior es:

- ***rJava***¹ [19]: definido en su página web como “*a simple R-to-Java interface*”, proporciona un puente de bajo nivel entre *R* y *Java* (vía *JNI*). Permite crear objetos, hacer llamadas a métodos y acceder a campos de objetos *Java* desde *R*.
- ***JRI*** [20]: definido en su web como “*a Java/R Interface, which allows to run R code inside Java applications as a single thread*”. Básicamente carga librerías dinámicas de *R* dentro de una aplicación *Java* y la dota de una *API* con la funcionalidad de *R*. Está apoyado por llamadas simples a *R* y mantiene en ejecución por debajo un entorno *R* completo (*REPL*).

Al igual que su competidor *RComander*, *JGR* también ofrece la posibilidad de extender sus menús con nuevas operaciones y herramientas. También presenta limitaciones para extenderlo, lo que no nos permite utilizarlo como punto de partida para desarrollar la aplicación *OBANSOft*.

Conclusión: hasta aquí el recorrido por las principales herramientas que podrían servir como base para comenzar un proyecto de la envergadura de *OBANSOft*. Se ha comprobado

¹Para completar aun más el análisis descrito en este documento, si estamos interesados en realizar llamadas a *C* podemos utilizar la interface *C/.Call C*.

que existe muy buen trabajo realizado respecto a dotar de interfaces gráficas de usuario al proyecto *R*. Sin embargo, estas interfaces se limitan a trabajar con los conceptos de estadística clásica dejando a un lado el área de interés de este trabajo: la estadística bayesiana. La existencia de un manual bastante completo [10] en lo que a análisis objetivos se refiere, se ve limitada por la inexistencia de herramientas que faciliten su difusión entre la comunidad científica no especializada en este campo de investigación. También, a la larga, los algoritmos de simulación utilizados en estadística bayesiana requerirán de necesidades de cómputo elevadas que merecerá la pena atacar con computación de altas prestaciones. Sería conveniente que la *GUI* implementada pudiese aprovechar este tipo de procesamiento haciéndolo transparente a los usuarios.

Por los motivos ya comentados, vamos a utilizar *R* como el motor de cómputo base de *OBANSOft* en la medida en que sea posible. Se perseguirá que no exista dependencia entre la interfaz gráfica y la librería estadística que contiene los algoritmos programados. Teniendo en cuenta esta separación, y por ser un lenguaje ideal para modelar escenarios con relaciones ricas entre sus elementos, vamos a utilizar *Java* para modelar la forma de trabajar en la estadística bayesiana. Además, utilizar *Java* permitirá encontrar una infinidad de paquetes para enlazarlo con otras tecnologías y arquitecturas. Por último, implementaremos la interfaz de usuario con los paquetes de objetos gráficos de *Java* (*Java Swing*) para que la aplicación sea multiplataforma además de tener un interfaz amigable, que es uno de los retos en un software que puede llegar a ser referente en cuanto a las funcionalidades que ofrece.

3 Diseño de la aplicación *OBANSOft*

En este apartado se comentarán los aspectos más relevantes a tener en cuenta para el diseño e implementación de la aplicación *OBANSOft*. En primer lugar, se resumirán los principales requisitos y funcionalidad que soporta la aplicación, así como los diferentes módulos que la componen. En segundo lugar, se explicarán las diferentes arquitecturas que integra *OBANSOft* para su funcionamiento. Por último, se comentará la posibilidad de extensión con nuevos modelos y algoritmos, y los diferentes recursos que se ofrecen a los usuarios finales.

3.1 Requisitos, funcionalidad y módulos

Después de unas primeras sesiones de extracción de requisitos compartidas con los compañeros estadísticos se concluyó que: para desarrollar una aplicación que infiera con modelos estadísticos bayesianos sobre bancos de datos, se debe soportar previamente la funcionalidad mínima para la edición y manipulación de los mismos. Esta funcionalidad mínima incluye también funciones que nos permitan extraer descriptivos sobre los datos para evaluar la calidad de los resultados.

En este apartado se especifican de modo general los requisitos extraídos que debe soportar la aplicación, y en conjunto darán una idea general de su funcionamiento. Se han

dividido los requisitos en módulos según las tareas que realizan. A lo largo del documento volveremos a referenciar a estos módulos para facilitar algunas explicaciones.

3.1.1 Creación y edición de bancos de datos (*DataFrame*)

- El objeto de datos principal de la aplicación es el *DataFrame*. Un *DataFrame* es una matriz de datos almacenados, normalmente, por columnas. Cada una de las columnas representa una variable y podrá ser de tipo: numérico, factor, ordenado, texto o lógico.
- Las columnas y datos de un *DataFrame* pueden ser manipuladas: renombrar, modificar valor, transformación de tipos, etiquetar factor, eliminar columna y ordenar un *DataFrame* según una columna de tipo numérico.
- Se permitirá la inserción y eliminación de filas y columnas sobre un *DataFrame* ya creado. En la inserción de columnas, el número de elementos se corresponderá con el número de datos previamente cargados en dicho *DataFrame*.
- La modificación de un valor en un *DataFrame* debe ajustarse a las restricciones del tipo de columna a la que pertenece.
- Se podrán crear objetos *DataFrame* vacíos.
- La aplicación define un directorio de trabajo que podrá ser modificado por los usuarios. Los resultados obtenidos serán guardados por defecto en dicho directorio.
- Para cualquier elemento de un *DataFrame* se debe permitir la manipulación del valor no asignado (“NA”), representando carencia de datos. En estadística esta situación es muy común. Las nuevas filas y columnas introducidas contendrán en sus celdas el valor “NA”.
- Se podrán cargar objetos *DataFrame* almacenados previamente en otros formatos: fichero de texto plano delimitado, fichero *Microsoft Excel* (“.xls”), fichero en formato *SPSS portable datafiles* (“.por”) y objetos *R* (“.Rdata”).
- Un *DataFrame*, ya sea el frame completo o una fracción del mismo, se podrá exportar a los formatos: *Ascii*, *Microsoft Excel* (“.xls”), objetos *R* (“.Rdata”), código *HTML* y tablas en código *LaTeX*.
- La aplicación debe permitir la carga simultánea de múltiples bancos de datos identificados por su nombre. No se permitirán dos *DataFrames* con el mismo nombre.
- Sobre un *DataFrame* se permiten operaciones de: copiar, pegar, eliminar, ocultar, renombrar y mezclar.

3.1.2 Transformaciones sobre los *DataFrame*

- Sobre las columnas de los *DataFrames* se deben permitir operaciones de transformación entre ellos. Por ejemplo, combinación de los datos de dos columnas o operaciones sobre una de ellas.
- Las operaciones permitidas son: operaciones básicas (suma, resta, multiplicación y división), funciones elementales (raíz, exponencial, logaritmo...) y por último, introducción de ruido aleatorio generado por las distribuciones: Uniforme(0,1), Exponencial(), Normal(), Student() y Cauchy().

3.1.3 Simulación de datos

- Se podrán generar columnas utilizando simuladores de distribuciones de probabilidad.
- Cada simulación está identificada por su nombre y parametrización.
- Para cada simulación se muestra una ayuda con la parametrización que ha sido implementada.
- El programa solicita por pantalla los parámetros a conocer para cada simulación que esté seleccionada. Además, adapta los formularios de manera que solo solicite y muestre la información de la distribución que queramos simular. Controla que los valores introducidos son correctos y cumplen las condiciones.
- La programación de los algoritmos de simulación debe recoger y comprobar las condiciones de los parámetros introducidos.

3.1.4 Descriptivas numéricas y gráficas

- La aplicación permite la elaboración de descriptivas numéricas y gráficas sobre las columnas de los *DataFrames*.
- Se implementan descriptivas tanto univariantes como multivariantes.
- Los tipos de las descriptivas dependen del tipo del dato sobre el que queremos realizarlas. Se debe reconocer el tipo de la variable (columna) seleccionada y adaptar las opciones de la descriptiva a ella. Además, los usuarios pueden configurar estas opciones según sus necesidades.
- Los resultados de las descriptivas numéricas son mostrados en tablas. Estas tablas deben incluir el resultado de todas las opciones seleccionadas.
- Los resultados de las descriptivas gráficas son devueltos como gráficos. Se ofrece la posibilidad de obtenerlos en formato *JPG* y *PNG*.

3.1.5 Inferencia bayesiana y modelos

- La aplicación soporta la inferencia a través de siguiente modelos bayesianos: Uniforme, Bernoulli, Binomial, Poisson, Exponencial y Normal (con media o varianza conocida).
- De manera similar a los algoritmos de simulación, la información sobre la parametrización utilizada se ofrece a través de la Ayuda. La parametrización del modelo es determinante a la hora de programarlo y debe ser explícita para el usuario.
- La inferencia con estos modelos utiliza a su vez los algoritmos de simulación mencionados en los puntos anteriores.
- El resultado de la inferencia con cada modelo (simulaciones de las distribuciones posterior y/o predictiva) será representado como un nuevo *DataFrame* de una única columna. El número de elementos de la columna resultado será igual al número de simulaciones configuradas para la ejecución del modelo.
- La aplicación debe recoger y solicitar al usuario los parámetros que sean necesarios para el modelo seleccionado. Cada modelo tiene unos parámetros y particularidades que son recogidos por la aplicación.
- La programación de los algoritmos debe recoger y comprobar las precondiciones de los parámetros introducidos.

Los requisitos enumerados han sido extraídos después del análisis de las aplicaciones citadas en el Apartado 2. Se ha intentado integrar lo mejor de las distintas aplicaciones analizadas, pero adaptándolas a las características de la estadística bayesiana.

Cabe destacar que estos requisitos están enunciados de manera general para entender el objetivo de la aplicación. Cada uno de ellos se divide a su vez en otros detalles que ha sido necesario tener en cuenta.

3.2 Aspectos más relevantes del diseño

Se ha desarrollado la aplicación utilizando un lenguaje de orientación a objetos que nos permita modelar las entidades extraídas a partir de los requisitos. Se ha utilizado programación en *Java*.

3.2.1 Aplicación del modelo Modelo-Vista-Controlador (*MVC*)

Para minimizar el acoplamiento entre el entorno gráfico y el modelo estadístico, se ha utilizado en el diseño de la aplicación el modelo Modelo-Vista-Controlador (*MVC*). Será independiente la lógica del modelo estadístico de la lógica de la interfaz de usuario. En la Figura 1 se observa la composición de los tres elementos del modelo *MVC*. Entre paréntesis se especifica la tecnología utilizada en cada uno de ellos.

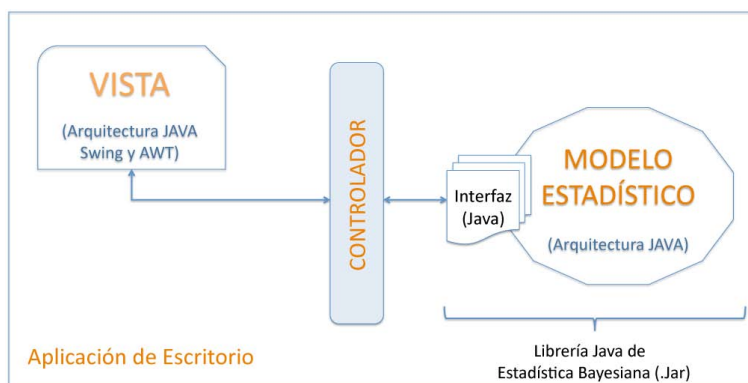


Figura 1. Diagrama general de la arquitectura de la aplicación OBANSOft.

Si se establece la correspondencia entre los requisitos del apartado anterior y los elementos la Figura 1 quedaría como:

- **Modelo:** define el modelo estadístico bayesiano que constituye el núcleo operacional de la aplicación. Las entidades extraídas de los requisitos son diseñadas e implementadas como objetos *Java*. Estos objetos definen sus interfaces de manera que faciliten su interacción. El modelo completo se utiliza para construir una librería *Java* (*.jar*) que aglutine toda la funcionalidad definida e implementada.
- **Vista:** es la extensión gráfica del modelo. La representación gráfica de la librería *OBANSOft* ha sido implementada como una aplicación de escritorio. La vista es el conjunto de los formularios, eventos, listener... y demás objetos que dotan al modelo de mayor interactividad.
- **Controlador:** sirve de enlace entre el modelo y la vista. Reduciendo al máximo el acoplamiento entre ellos se facilitará el futuro despliegue del modelo en otros entornos y arquitecturas (por ejemplo, aplicación web).

Al utilizar el modelo *MVC*, se debe establecer desde un inicio la separación entre las competencias del Modelo y las competencias de la Vista. Para no sobrecargar este documento, se va a comentar un ejemplo representativo sobre esta división del trabajo: imaginemos el caso de la carga de un nuevo *DataFrame* en la aplicación. El ciclo de vida natural de esta acción sería:

1. Se indican el número de columnas y tipos del nuevo *DataFrame*.
2. Se indican el número de filas.
3. Creamos un nuevo objeto *DataFrame* en el Modelo.
4. Lo registramos como un nuevo frame en la aplicación.
5. Cargamos y mostramos el *DataFrame* creado en los formularios gráficos de la aplicación.

La pregunta que debemos plantearnos es, ¿quién debe encargarse de hacer cada uno de los pasos? Si se carece de experiencia podríamos construir una única clase encargada de hacer todos los puntos secuencialmente, sin embargo se estarían incumpliendo los principios del modelo *MVC*. De esta manera, quedaría mezclada la lógica de la Vista con la lógica del Modelo suponiendo un lastre para una futura adaptación del modelo a una Vista en otra tecnología. Para seguir el modelo *MVC*, se identifican objetos en cada uno de los tres ámbitos que ejecutan los pasos correspondientes: objetos del Modelo (paso 3), objetos de la Vista (pasos 1-2-4) y objetos del Controlador (paso 5 y es el que coordina a los objetos del modelo y la vista entre los pasos 2-3 y 3-4).

Se comenta el uso del modelo *MVC* en la aplicación porque ha sido un punto decisivo en la implementación de *OBANSof*t. En algunos de los módulos estaba claro de quien era competencia hacer unas operaciones u otras, y por tanto definir las interfaces y su interacción no ha supuesto problemas. Sin embargo en otros módulos ha resultado más complejo detectarlo, por ejemplo en: la validación de nuevos valores en los campos del *DataFrame*, la simulación de nuevas columnas y, sobre todo, en el módulo de los modelos de inferencia bayesiana (por existir varios tipos de análisis para cada modelo y, a su vez, parámetros particulares de cada tipo de análisis). Definir correctamente estas competencias ha sido decisivo para un mejor funcionamiento de la aplicación. En los casos dudosos, los errores han supuesto desviaciones significativas en el tiempo planificado para su implementación.

3.2.2 Principales objetos e interacción entre ellos

Se comentará en esta subsección cuales son los objetos principales que componen *OBANSof*t, y se comentan los aspectos más interesantes.

Objetos del Modelo

Se pretende ilustrar con la figura Figura 2 cuales son los principales objetos que componen el Modelo de la librería *OBANSof*t. El objeto principal y más complejo es el objeto *DataFrame*. En él se centraliza todo lo referente a bancos de datos.

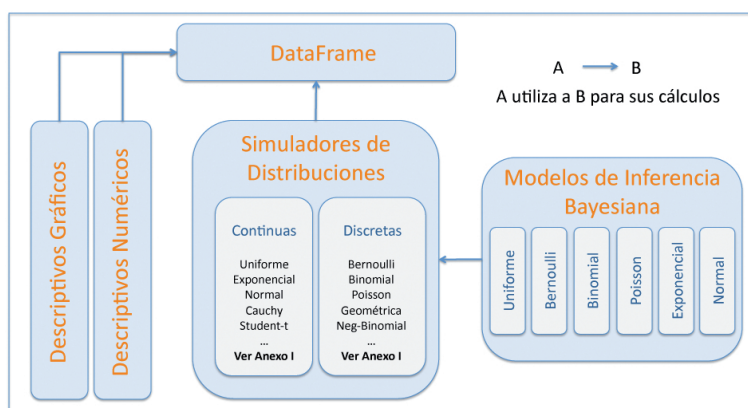


Figura 2. Principales objetos que forman el modelo estadístico.

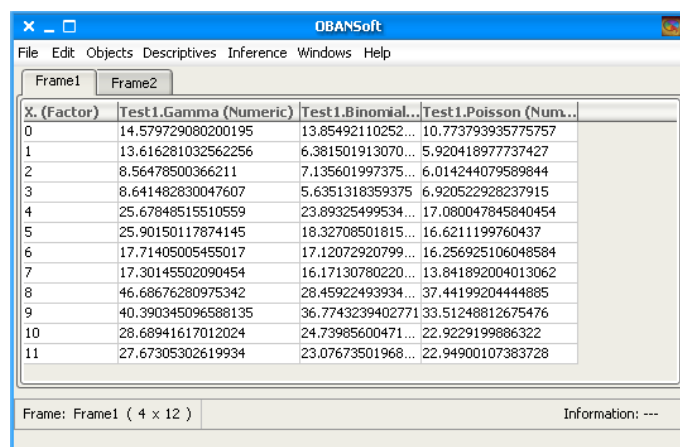
Como ya se ha comentado en el apartado anterior, uno de los problemas principales con el que nos hemos encontrado es la cantidad de diferencias entre elementos de un mismo módulo. Por ejemplo las diferencias entre las opciones de las descriptivas gráficas en función del tipo de datos; o entre las parametrizaciones de los modelos y los simuladores. En estos casos, se han utilizando las posibilidades de herencia que ofrece *Java* para modelarlo. Se ha definido la funcionalidad básica que debe soportar la clase más general, y se ha heredado de ella en las clases hijas adaptando los parámetros y opciones a cada caso particular.

A modo ilustrativo, en la Figura 2 todos los objetos utilizan al objeto *DataFrame* para su funcionamiento. Se ha procurado no crear dependencia entre los distintos objetos, y delegar esta interacción en el controlador que le corresponda. Así cada objeto se define con interfaces con datos generales, y será el controlador el encargado de mapear entre los datos de unos y otros para conseguir el objetivo deseado. Si ejemplificamos de nuevo esta situación, las descriptivas numéricas se realizan sobre variables columna de un *DataFrame*. Para no crear la dependencia, los objetos descriptivas no esperan una columna de un frame sino un array de elementos. Es el controlador el que se encarga de localizar el Frame, solicitar la columna y entonces pasarla a la operación de la descriptiva.

Objetos de la Vista

La Vista estará formada por objetos gráficos (librería *Swing*) que extienden su funcionalidad haciendo uso de los objetos del modelo. Por poner algunos ejemplos, destaca el uso del *JTable* para representar los *DataFrames*, el uso de *JTabPannel* para agregar los diferentes *Frames* cargados simultáneamente o el uso de los fondos de los *JPanel* para cargar las imágenes de las descriptivas gráficas.

El usuario manipulará directamente los formularios de la Vista, y esta se comunicará con el controlador correspondiente, que será el encargado de invocar al modelo y devolver a la vista los resultados. En la Figura 3 se ilustra el formulario principal (“*MainForm*”) desde el que se puede acceder a toda la funcionalidad disponible.



X. (Factor)	Test1.Gamma (Numeric)	Test1.Binomial...	Test1.Poisson (Num...
0	14.579729080200195	13.85492110252...	10.773793935775757
1	13.616281032562256	6.381501913070...	5.920418977737427
2	8.56478500366211	7.135601997375...	6.014244079589844
3	8.641482830047607	5.6351318359375	6.920522928237915
4	25.67848515510559	23.89325499534...	17.080047845840454
5	25.90150117874145	18.32708501815...	16.6211199760437
6	17.71405005455017	17.12072920799...	16.256925106048584
7	17.30145502090454	16.17130780220...	13.841892004013062
8	46.68676280975342	28.45922493934...	37.44199204444885
9	40.390345096588135	36.7743239402771	33.51248812675476
10	28.68941617012024	24.73985600471...	22.9229199886322
11	27.67305302619934	23.07673501968...	22.94900107383728

Figura 3. Formulario principal de la aplicación.

El “*MainForm*” se comporta como un contenedor de *DataFrames* (en el ejemplo de la Figura 3 hay cargados dos *frames* con nombre “Frame1” y “Frame2”) y de los resultados de otras operaciones (descriptivas, simulaciones e inferencias). En la parte superior de la Figura 3 se observa el menú principal, que dispone de secciones individuales para los módulos identificados en las sesiones de requisitos. A través de estos menús invocaremos a las distintas operaciones que se ejecutarán sobre el frame activo en ese momento.

Objetos del Controlador

Se ha definido un controlador principal que gestiona los eventos que requieren la participación del “*MainForm*”. Para el ejemplo anterior de creación de un *DataFrame* este era el controlador referido (“*MainController*”). Además, para favorecer la extensión de cada módulo (tras esta primera versión) con nueva funcionalidad, se ha definido un controlador para cada uno de estos módulos. En *OBANSoft* tenemos definidos un total de cuatro: “*FileController*”, “*EditController*”, “*DescriptiveController*” e “*InferenceController*”.

Además de estos controladores, la existencia de dos objetos con una lógica compleja requerirá el esfuerzo de definir su propio controlador. Por un lado, definiremos un controlador para los *DataFrames* que se encuentren cargados en la aplicación (“*DataFramesController*”). También será necesario un controlador para los objetos de las descriptivas gráficas, interesa mantener cargadas simultáneamente tantas como se desee y será necesario definir un elemento que lo controle.

3.2.3 Algoritmos bayesianos. Integración de tecnologías

Hasta este punto del documento se han comentado aspectos generales de la aplicación que modelan el procedimiento de trabajo en estadística bayesiana. Aunque se ha hablado de simuladores y modelos estadísticos, no se ha mencionado nada sobre los algoritmos que utilizamos para ello.

Poder utilizar librerías y funciones ya programadas en *R* permite ahorrar tiempo en su implementación y centrarnos más en su paralelización. También se podrá adelantar la implementación de algoritmos *MCMC*, en muchos casos no implementados, y que consumen tiempos de ejecución desorbitados. Cabe destacar que el objetivo principal de la implementación de *OBANSoft* es poder utilizarlo en el futuro para integrar este último tipo de algoritmos paralelizados. Por este motivo, cobra especial importancia diseñar la aplicación para facilitar la integración de algoritmos más complejos.

Estos nuevos algoritmos se programarán en lenguaje *C* y se paralelizarán para varias arquitecturas. Por ejemplo, implementar un algoritmo y paralelizarlo con memoria compartida, memoria distribuida y *GPU*. Deberíamos integrar en la aplicación el algoritmo paralelo para un tipo de arquitectura, y que el usuario final solo perciba una ganancia en tiempo o calidad de las simulaciones. Otra opción será utilizar la combinación de *R*, *C* y *Java* para poder aprovechar funciones ya programadas en alguno de estos lenguajes.

La convivencia de algoritmos en diferentes arquitecturas supone la ampliación de la Figura 1 en la parte del Modelo Estadístico. Concretamente, se define una interfaz para

cada algoritmo que describe todos los parámetros y funciones que lo componen. Las llamadas a estos algoritmos se centralizan en una clase principal que actúa como fachada. Cualquier objeto del modelo que requiera utilizar algunos de estos algoritmos lo invocará a través de la fachada. Son los métodos de la fachada los encargados de: o implementar directamente el algoritmo deseado o utilizar las librerías necesarias para invocar a los algoritmos programados en otros lenguajes. El diseño de esta integración se ilustra en la Figura 4.

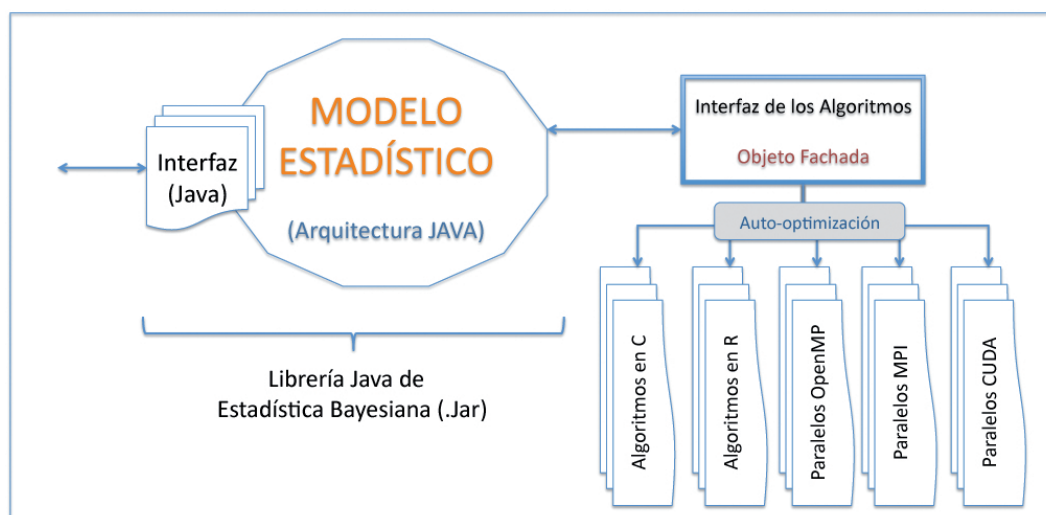


Figura 4. Integración de diferentes tecnologías para la implementación de los algoritmos estadísticos.

Este diseño favorecerá que a medida que vayamos paralelizando los algoritmos o optimizándolos podamos incluirlos en la aplicación simplemente modificando el enlace del objeto fachada. Otro aspecto importante es que cada uno de estos algoritmos puede estar haciendo uso de una arquitectura de computador diferente. En caso de distribuir *OBANSoft* como una aplicación de escritorio, puede ser que el equipo donde se instale no disponga de todas estas arquitecturas. En este caso, debemos incluir en la fase de instalación (véase en la Figura 4) un algoritmo de autoconfiguración que sea capaz de detectar que versiones de los algoritmos se pueden ejecutar en dicha máquina y configurar la fachada para invocar a los posibles.

3.2.4 El R-Modelo y su integración con *R*

Como ya se ha comentado en apartados anteriores, en esta primera versión de *OBANSoft* desarrollada durante esta tesis de máster se ha investigado la manera de integrar el modelo *Java* con una instancia de *R*. En el Anexo I se encuentran las equivalencias entre las funciones de simulación y transformación definidas en el Modelo estadístico y sus respectivas llamadas *R* (en caso de existir).

El enlace entre *Java* y *R* lo realizamos con la librería *JRI*. A través de la conexión que abrimos con *JRI*, podremos invocar la ejecución de funciones en *R*. Esta conexión que se abre (*rEngine*) permanecerá activa hasta que se invoque explícitamente su finalización;

es decir, cualquier objeto creado desde la ejecución de *Java* se mantendrá cargado hasta que se finalice la conexión.

En la Figura 5 se tiene el objeto *Java* R-Modelo. El R-Modelo es el encargado de almacenar todos los algoritmos programados en *R*. Extiende la interfaz de los algoritmos que se encuentra definida en el Objeto Fachada, pero utiliza el *rEngine* para ejecutarlos. Una vez utilizado *R* para las operaciones que interese, obtenemos los resultados convirtiéndolos a objetos *Java* que devolvemos de nuevo al modelo estadístico a través de la fachada.

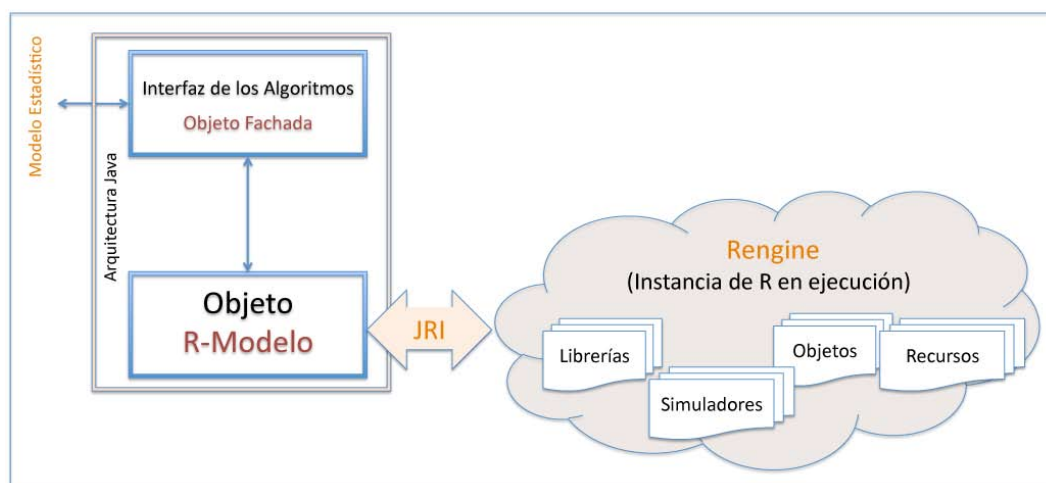


Figura 5. El objeto *Java* R-Modelo y su conexión con *R* por medio de la librería *JRI*.

Se propone seguir una metodología similar a esta para extender *OBANSOft* con nuevos algoritmos paralelizados. A medida que se utilicen otros lenguajes, deberemos encontrar una librería para enlazarlos a *Java*.

3.2.5 El log de la aplicación y otras cuestiones

La aplicación permite habilitar un fichero de log que registra todas las operaciones que suceden durante la ejecución de una sesión del programa. Se han programado dos logs con diferentes ámbitos: un log que registra todas las acciones que involucran a objetos de la Vista y otro log que registra cualquier operación solicitada al R-Modelo. Se decidió implementar estos bitácoras para que sea más sencillo hacer un seguimiento de los errores que se produzcan durante la fase de pruebas con usuarios finales.

Por último, sería interesante destacar que la carencia de una documentación más elaborada y extensa respecto al diseño de la aplicación (utilizar diagramas y notaciones UML) se debe a que no hemos querido sobrecargar la memoria con información que pueda resultar confusa. Además, la intención de esta tesis de máster no es centrarnos únicamente en el desarrollo de la aplicación sino avanzar al análisis y paralelización de los algoritmos; por este motivo, hemos decidido avanzar en esta línea y no detenernos demasiado en otros aspectos del programa.

3.3 Recursos disponibles y futura extensión

Con este apartado se quiere puntualizar que recursos quedan disponibles para que otros usuarios puedan utilizar en sus prácticas estadísticas. Los usuarios finales podrán solicitar cualquiera de estos dos recursos: la **aplicación de escritorio** completa o la **librería Java** que contiene únicamente las operaciones utilizadas. Esta librería se descargará como un fichero *.jar* que podrá ser importado desde otros proyectos *Java*. Como la librería utiliza el motor de *R* por debajo, los usuarios deberán tener instalada al menos una versión de *R* que permita ser configurada con el enlace.

Como tercera opción, se encuentra instalada la última versión de la aplicación en uno de los servidores del *GCCPPUM*. Se ha configurado el acceso a través de una cuenta remota con acceso *SSH* habilitada para los usuarios que deseen probar puntualmente el programa sin necesidad de tener que desplegar todo el entorno en sus equipos.

Respecto a la posibilidad de **extensión de la aplicación**, se ha comentado que el diseño previo favorece la facilidad de añadir nuevas operaciones y modelos más complejos. A fecha de la terminación de esta tesis de máster, queda pendiente el desarrollo de un manual explicativo de la extensión de *OBANSofT*. No obstante el **código fuente estará disponible** para los usuarios que deseen extenderlo y colaborar. Deberá solicitarse a la dirección de correo del grupo *GCCPPUM* (pcgumu@gmail.com).

4 Optimización y paralelización de los algoritmos

Una vez finalizada la primera versión de la aplicación se ha procedido (como se explicó en la metodología del apartado 1.2.2) a estudiar el rendimiento de los algoritmos programados. Hasta el comienzo de este apartado, solo se habían programado versiones secuenciales del código. Se pretende desde este punto, estudiar el funcionamiento de los algoritmos y los recursos consumidos. Tras ello, se seleccionarán aquellos que consuman mayor tiempo de ejecución y se estudiarán para paralelizarlos en entornos de memoria compartida. Con este tipo de paralelización, los usuarios que instalen la aplicación podrán aprovechar la arquitectura de los procesadores *multicore*. El crecimiento entre los usuarios medios de este tipo de procesadores ha aumentado enormemente en los últimos dos años.

4.1 Estudio del rendimiento de los algoritmos de simulación

Por ser los algoritmos más complejos, vamos a centrarnos en el estudio de los simuladores de distribuciones. Además, estos algoritmos son utilizados para la inferencia de los modelos bayesianos, y por tanto optimizarlos supondrá mejoras en su funcionamiento.

Como se comentó previamente, para cada algoritmo invocamos su función en *R*, de la que podemos conocer su código fuente o no. Como no se ha encontrado el código fuente de todas las funciones utilizadas, independientemente de la función con la que se trabaje,

se van a considerar como cajas negras de las que no se conocen más detalles sobre su implementación.

El objetivo del Experimento 1 será estudiar y comparar los tiempos de ejecución de los simuladores (en total 27 algoritmos). Para cada distribución simulada, se repetirá su ejecución variando los tamaños en el intervalo [1.000.000,20.000.000] incrementando en 1.000.000. Obtendremos los tiempos de ejecución en mseg de cada simulación para distintos tamaños. Adicionalmente, para obtener valores de los tiempos más fiables se ha repetido el experimento 10 veces y calculados valores promedios de las 10 ejecuciones.

Se pretende determinar con este primer experimento la tendencia del tiempo de ejecución a medida que aumentamos el tamaño de las simulaciones. De los resultados obtenidos se concluye que el tiempo consumido crece linealmente en función del tamaño de las simulaciones. En la Figura 6 se muestra el crecimiento para 5 simulaciones representativas.

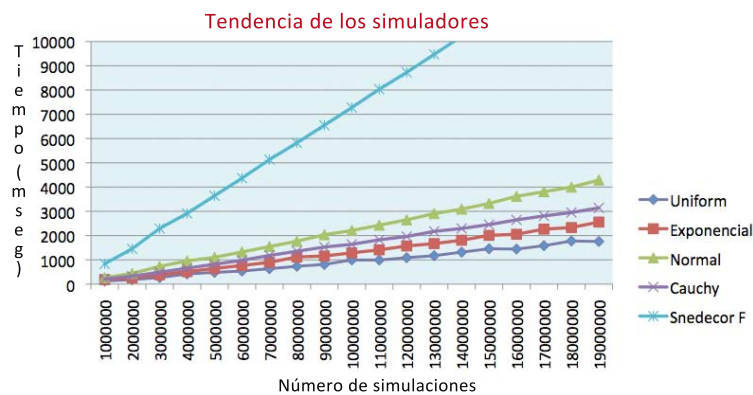


Figura 6. Tiempos de ejecución aumentando el tamaño de las simulaciones. Muestra representativa de 5 de las 27 distribuciones disponibles.

Sin embargo a pesar de seguir la misma tendencia, el tiempo de ejecución varía según la distribución simulada. Para estudiar esta diferencia, se agrupan los datos del Experimento 1 de la siguiente manera: aprovechando el crecimiento lineal de todos los simuladores se obtiene para cada simulador el tiempo medio para tamaño un millón (Figura 7).

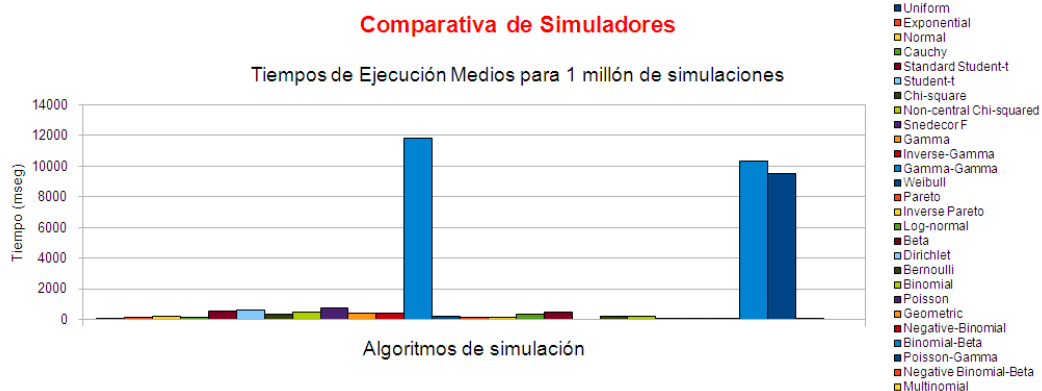


Figura 7. Gráfico comparativo resultado de utilizar todos los simuladores disponibles en la aplicación.

De los tiempos de la Figura 7, podemos concluir que para tamaños de simulaciones pequeños el tiempo de ejecución es inapreciable. El verdadero problema de los algoritmos de simulación viene cuando se utilizan para inferir en los modelos bayesianos. En estos modelos se lanzan simulaciones continuamente hasta que el algoritmo converge a un cierto valor o a pasado un tiempo prefijado. Es en estos modelos cuando el tiempo de convergencia puede oscilar desde unos pocos minutos, hasta horas, días o incluso meses en los casos multivariantes. La paralelización de los simuladores podría reducir el tiempo de convergencia o que se alcancen mejores soluciones en el mismo tiempo. En la Figura 8 se ha repetido el gráfico de la Figura 7 pero eliminando los tres simuladores más lentos, que impedían apreciar las diferencias entre los demás algoritmos.

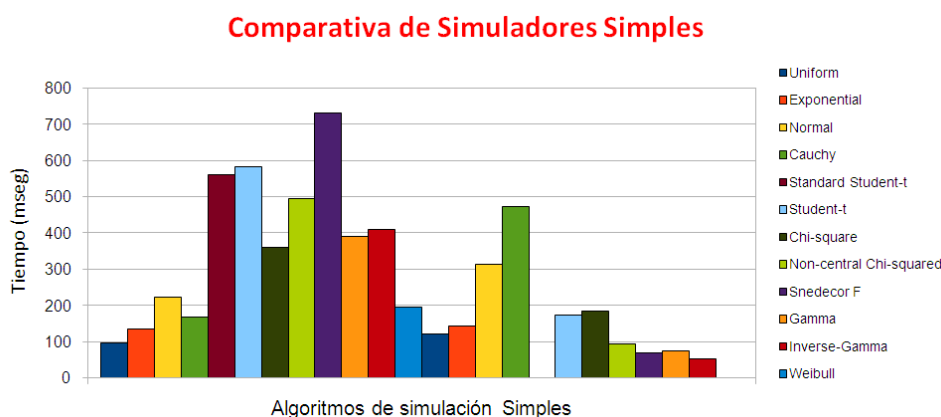


Figura 8. Gráfico comparativo con los tiempos medios de ejecución de las funciones simples.

Analizando los valores aproximados de los tiempos de la Figura 7, podemos dividir los algoritmos de simulación en dos grupos. Por un lado están las simulaciones que consumen menos de medio segundo en su ejecución que las llamaremos funciones simples. Por otro lado, a las tres funciones que consumen mayores tiempos de ejecución las nombraremos funciones compuestas. La explicación a esta diferencia de comportamiento se obtiene a partir del análisis de los códigos fuente que se han podido localizar:

- Las funciones simples invocan a funciones de R con una llamada similar a:

```
rFuncion("TamSimulaciones", ... Otros parámetros ...)
```

Internamente este algoritmo utiliza el parámetro "TamSimulaciones" para generar un array con ese número de componentes, que almacena el resultado de la simulación. El resultado de la simulación dependerá del tamaño de las simulaciones, es decir, no se obtendrán los mismos resultados al ejecutar `rFuncion(50, ...)` que ejecutando 50 veces `rFuncion(1, ...)`.

- El algoritmo de las funciones compuestas es completamente diferente. En primer lugar se invoca la simulación de tamaño "TamSimulaciones" de una función simple. El resultado obtenido ("result") será utilizado como parámetro de otra función

simple, que llamaremos función cadena. Sin embargo, la llamada a la función cadena será del tipo `rFuncion(1, "result[i]")` y se invocará “TamSimulaciones” veces.

En el Código 1, se muestra el código *R* para implementar el algoritmo de la función compuesta *Gamma-Gamma*. En este caso particular, coincide la función de simulación para la función simple y la función cadena.

```

1      rgamma.gamma = function(nsim, alpha, beta, nu) {
2          theta=rgamma(nsim, alpha, beta);
3          x=vector(length=nsim);
4          for(i in 1:nsim) {
5              theta[i]=rgamma(1, nu, theta[i]);
6          };
7
8          return(theta);
9      }

```

Código 1. Algoritmos de simulación de la función compuesta *Gamma-Gamma*.

Es la composición de dos o más funciones de simulación lo que provoca que los tiempos de ejecución de las compuestas sean muy superiores a las simples.

4.2 Introducción del paralelismo

A partir del estudio secuencial de los simuladores, y por ser los que mayores tiempos de ejecución consumen, afrontaremos la paralelización de las tres funciones compuestas. Además, su comportamiento es similar al que tendrán los algoritmos *MCMC* que serán los siguientes a implementar como continuación de este trabajo de tesis de máster.

Puesto que estos tres algoritmos están implementados en código *R*, se han estudiado las posibilidades de utilizar *R* para paralelizar código en entornos de memoria compartida. Cabe destacar que esta solución no ha sido utilizada hasta la fecha por ningún investigador del grupo GCCPPUM y no sabemos a priori como se comportarán estas librerías.

Con el objetivo de determinar el peso en el tiempo total de ejecución de la función simple y la función cadena, se ha llevado a cabo el Experimento 2. Se ha repetido el experimento 10 veces, y lanzado cada uno de los tres simuladores con tamaños comprendidos en el intervalo [1.000.000, 5.000.000] incrementado en 1.000.000. Al igual que en la última parte del Experimento 1 ha obtenido el tiempo medio para tamaños de 1.000.000. Como consecuencia de este experimento hemos obtenido el porcentaje del tiempo total consumido por la función simple y la compuesta (Figura 9).

De los resultados del Experimento 2 podemos concluir que con la paralelización y disminución del tiempo invertido en la función cadena, conseguiríamos reducir el tiempo total por suponer la parte paralelizada un porcentaje mayor del 90% del tiempo de ejecución.

El problema ahora es decidir cómo podríamos modificar el algoritmo del Código 1 para convertirlo en una solución paralela, manteniendo la validez de los resultados. Al repetir la simulación de la función cadena dentro de un bucle de tamaño “TamSimulaciones”, podríamos lanzar la ejecución del bucle en paralelo, por tratarse de simulaciones

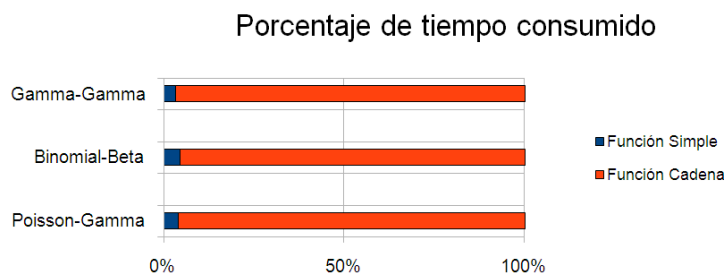


Figura 9. Porcentajes de tiempo consumidos en la ejecución de una función compuesta por la función simple y la función cadena.

independientes. De esta forma el paralelismo no participará en la ejecución de la primera función simple, pero su tiempo es despreciable frente al de la función cadena.

Entre las diferentes librerías que hemos encontrado, hemos decidido utilizar *SnowFall* que permite paralelizar código utilizando programación en *R*. Esta librería funciona de forma similar a *OpenMP* de *C*. La forma de paralelizar código con esta librería consiste en definir un núcleo de computación (*wrapper*) que posteriormente se invocará a su ejecución en paralelo. Dependiendo del número de procesadores que pongamos en funcionamiento, se lanza en cada uno de ellos la ejecución del código implementado en el *wrapper*, estas ejecuciones serán independientes y no compartirán memoria del programa principal a menos que lo indiquemos con directivas. Mostramos el ejemplo de la función *GammaGamma* en el Código 2.

```

1      # Calculariamos con la funcion simple los parametros
2      # alpha y beta que usaremos en la funcion rgamma.
3      ... ..
4      library(snowfall)
5
6      # 1. Inicializamos snowfall
7      sfInit(parallel=TRUE, cpus=1, type="SOCK")
8
9      # 2. Cargariamos bancos de datos que queremos que
10     # sean leidos por todos los procesadores
11     # require(mvna)
12     # data(sir.adm)
13
14     # 3. Definimos el wrapper, el cual va a ser paralelizado.
15     wrapper <- function(idx) { return(rgamma(1,mu,theta)); }
16
17     # 4. Exportariamos los datos y paquetes que queremos
18     # que sean leidos por todos los procesadores
19     # sfExport("sir.adm")
20     # sfLibrary(cmprsk)
21
22     # 5. Inicializamos el generador paralelo de numeros aleatorios
23     sfClusterSetupRNG()
24
25     # 6. Distribuimos los calculos
26     result <- sfLapply(1:tamSimulaciones, wrapper);
27
28     # 7. Detenemos snowfall
29     sfStop()
30     ... ..
31     ## Devolvemos el resultado de la simulacion

```

Código 2. Algoritmo paralelo de la función cadena del simulador *Gamma-Gamma*.

Como se muestra en el Código 2, se define la función cadena *Gamma* dentro del *wrapper* (línea 15), y se ejecuta el bucle que la calcula en paralelo utilizando la función *sfLapply* (línea 26). Antes de la ejecución del *wrapper* deberemos indicar cuantas *CPUs* queremos utilizar con la función *sfInit* (línea 7). Otro aspecto a tener en cuenta es que en los algoritmos de simulación se utiliza la generación de números aleatorios. Cuando se paraleliza código, esta generación puede plantear problemas por suponer un cuello de botella entre los distintos procesos (estos bloqueos convertirían al algoritmo paralelo en un algoritmo parecido al secuencial). Para evitar este problema, hemos tenido que utilizar la función *sfClusterSetupRNG* (línea 23) que se encarga de distribuir las semillas en los distintos hilos para que el funcionamiento sea el adecuado.

4.3 Estudio experimental del paralelismo

En este último subapartado vamos a comentar los primeros resultados obtenidos ejecutando las versiones paralelas de los algoritmos de simulación compuestos. En el Experimento 3 se va a lanzar en paralelo la ejecución de simulaciones de tamaño 1 en el intervalo [1.000.000,3.000.000] incrementando en 500.000. Se estudiará la ejecución en paralelo de las tres funciones cadena *Gamma*, *Binomial* y *Poisson*. Repetiremos el experimento 5 veces y obtendremos los tiempos de ejecución medios entre todas ellas. En la Figura 10 se muestra la gráfica de los tiempos de ejecución medios al ejecutar el Experimento 3 en la máquina Luna que tiene un procesador *multicore* con 4 núcleos en total (ver más características en apartado de herramientas).

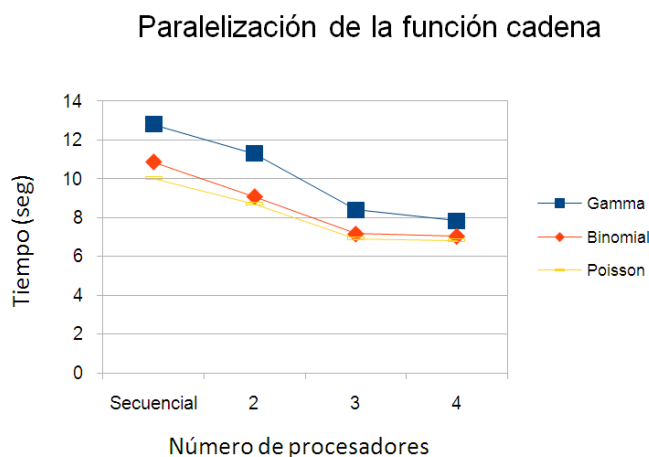


Figura 10. Resultados de ejecutar el bucle de la función cadena variando el número de procesadores entre uno y cuatro.

Las simulaciones de tamaño uno que se ejecutan en paralelo son independientes entre sí, y por tanto teóricamente se podría conseguir una reducción del tiempo de ejecución en un factor de p (donde p es el número de núcleos). Sin embargo, de los resultados de la Figura 10 las reducciones conseguidas no alcanzan este factor. Aunque se produce un pequeño descenso en tiempo de ejecución cuando pasamos de utilizar 1 a 2 procesadores, y en la misma proporción se reduce cuando utilizamos 3, esta ganancia está muy lejos del

límite teórico. Además, se observa un comportamiento todavía más extraño al permanecer fijo el tiempo de ejecución ya utilicemos 3 ó 4 procesadores.

Que el tiempo de ejecución permanezca constante al pasar de 3 a 4 núcleos se puede justificar porque, aunque utilicemos un núcleo más, por la arquitectura de Luna estos núcleos se dividen en dos chips; y por tanto cada dos núcleos comparten memoria caché. Al aumentar de 3 a 4 núcleos, asignamos más trabajo a un último core que está en el mismo chip, por tanto podrían incrementarse los fallos en caché y perjudicar así al tiempo de ejecución. Sin embargo, ¿por qué sí se produce un descenso cuando pasamos de 1 núcleo a 2? En este caso el procesador no está completamente ocupado y quedan núcleos libres de trabajo. La librería *SnowFall*, concretamente con su función *sfInit* podría no estar gestionando los hilos correctamente. Podría ser que al utilizar 2 *CPUs* no ejecute el trabajo distribuido siempre en los mismos núcleos sino que vaya variándolos, y de esta forma los fallos de caché descenderían.

Al ser la primera vez que trabajamos con esta librería, carecemos de referencias de otros problemas para poder concluir con certeza los motivos por los que nos alejamos de la ganancia teórica del problema. No obstante, merece la pena destacar, que pese a estos problemas este tipo de algoritmos sí parecen ser abordables con paralelismo. Aunque no se alcanza la reducción de tiempo teórica, con la paralelización se ha conseguido un descenso de tiempo de ejecución que podría suponer una ganancia significativa al utilizar el simulador en los algoritmos de inferencia con modelos.

5 Conclusiones y trabajos futuros

Hasta aquí se ha resumido todo el trabajo realizado dentro de esta tesis de máster. Como principales conclusiones tenemos que:

- La implementación de *OBANSoft* ha permitido cubrir el vacío existente dentro de las aplicaciones de estadística bayesiana. Integra los algoritmos de simulación y los modelos de inferencia (más básicos) que los expertos en el área pueden necesitar.
- El programa ha sido presentado en una sesión de póster del congreso “Valencia International Meeting on Bayesian Statistics, 2010 ISBA World Meeting”. Se realizó un póster interactivo en la sesión del pasado 5 de Junio de 2010, y tuvo gran aceptación sobre todo desde un punto de vista docente. Se puede consultar más información en el Anexo II.
- Por su diseño modular y por estar preparado para la integración de diversas tecnologías, tanto la librería como la aplicación completa van a servir de base para integrar otros resultados obtenidos en la continuación de esta línea de investigación. Entre otros logros, se perseguirá adaptar las paralelizaciones de los algoritmos *MCMC* para simplificar el uso del paralelismo a los usuarios finales.
- Basándonos en el estudio experimental de los algoritmos paralelos, parece que las funciones compuestas pueden ser paralelizadas para conseguir beneficios en el tiem-

po de ejecución. Sin embargo, utilizar la librería *SnowFall* ha limitado las conclusiones por no contar con más referencias sobre ella.

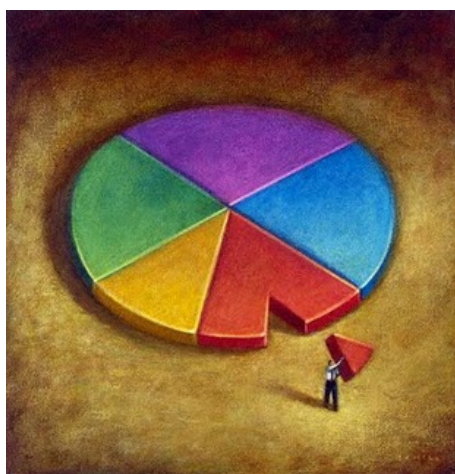
Se proponen como trabajos futuros y posibles líneas de investigación para continuar a partir de este trabajo:

- Continuar con el estudio del comportamiento de la librería *SnowFall*. Para asegurarnos que funciona correctamente podríamos aplicar la metodología seguida en el proyecto de Virginio García López [21]. De esta manera podríamos comprobar si se encuentra correctamente instalada y que rendimiento se consigue.
- Buscar los códigos de los simuladores programados en *C*; y en caso de no localizar ninguno, buscar directamente el algoritmo en pseudocódigo para programarlo utilizando este lenguaje. Una vez hecho esto, compararlos con los tiempos de ejecución al invocar a funciones de *R*.
- Paralelizar los algoritmos programados en *C* con *OpenMP* y comparar los resultados frente a la paralelización con *SnowFall*.
- Introducir los algoritmos *MCMC*, estudiar su paralelización y la aplicación de técnicas de autooptimización durante la instalación.
- Ampliar los módulos de *OBANSofT* con nueva funcionalidad y algoritmos.
- Adaptar el modelo estadístico para utilizarlo en una aplicación web.

Bibliografía

- [1] Parallel Computing Group, Universidad de Murcia. <http://www.um.es/pcgum/>.
- [2] J. Morales, M. E. Castellanos, A. M. Mayoral, R. Fried, and C. Armero. Bayesian design in queues: an application to aeronautic maintenance. *Journal of Statistical Planning and Inference*, 2007.
- [3] R. Zornoza, J. Mataix-Solera, C. Guerrero, V. Arcenegui, A. M. Mayoral, J. Morales, and J. Mataix-Beneyto. Soil properties under natural forest in Alicante province of Spain. *Geoderma* 142, 2007.
- [4] V. Arcenegui, J-Mataix-Solera, C. Guerrero, R. Zornoza, A. M. Mayoral, and J. Morales. Factors controlling the water repellency induced by fire in calcareous mediterranean forest soils. *European Journal of Soil Science*, 2007.
- [5] Dennis V. Lindley. The 1988 wald memorial lectures: The present position in bayesian statistics. *Statistical Science*, 5(1):44–65, 1990.
- [6] Berger. Bayesian analysis: a look at today and thoughts of tomorrow. *Journal of the American Statistical Association*, 2000.
- [7] The BUGS Project - Bayesian inference Using Gibbs Sampling. <http://www.mrc-bsu.cam.ac.uk/bugs/>.
- [8] James Berger. The case for objective bayesian analysis. *Bayesian Analysis*, 1:385–402, 2006.
- [9] Thomas H. Short. Subjective and objective bayesian statistics: Principles, models, and applications (2nd ed.). S. James Press. *Journal of the American Statistical Association*, 100:355–356, 2005.
- [10] R. Yang and J. O. Berger. A catalog on noninformative priors. *Discussion Paper, 97-42, ISDS, Duke University, Durham, NC*, 1996.
- [11] Kent Beck. *Extreme programming explained: embrace change*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
- [12] SnowFall. <http://cran.r-project.org/web/packages/snowfall/>.
- [13] SAS: bayesian capabilities in SAS/STAT Procedures. <http://support.sas.com/rnd/app/da/bayesproc.html>.
- [14] The R Project for Statistica Computing. <http://www.r-project.org/>.
- [15] R Commander: A Basic-Statistics GUI for R. <http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/>.
- [16] JGR - Java GUI for R. <http://jgr.markushelbig.org/JGR.html>.

- [17] John Fox. The R Commander: a basic-statistics graphical user interface to r. *Journal of Statistical Software*, Septiembre 2005.
- [18] Markus Helbig and Martin Theus. Jgr: Java gui for r. *Statistical Computing and Graphics*, December 2005.
- [19] rJava - Low-level R to Java interface - RForge.net. <http://www.rforge.net/rJava/>.
- [20] JRI - Java/R Interface - RForge.net. <http://www.rforge.net/JRI/>.
- [21] Virginio García López. Herramientas para el estudio de prestaciones en clusters de computación científica. aplicación en el laboratorio de computación paralela. Master's thesis, Universidad de Murcia, Febrero 2009.



UNIVERSIDAD DE MURCIA

ANEXO I

Aplicación OBANSoft

Listado resumen de operaciones estadísticas y
modelos bayesianos. Equivalencia con funciones de *R*

Autor:
Manuel Quesada Martínez

Apéndice de funciones soportadas por *OBANSoft*

1.1 Operaciones de transformación

En nuestra librería vamos a permitir operar entre las diferentes variables que conforman nuestro frame. Para ello vamos a detallar con más detalle de que operaciones se va a componer nuestra librería:

- SÍMBOLOS BÁSICOS -

Suma	+	$\text{frame\$V1} + \text{frame\$V2}$
Resta	-	$\text{frame\$V1} - \text{frame\$V2}$
Multiplicación	*	$\text{frame\$V1} * \text{frame\$V2}$
División	÷	$\text{frame\$V1} \div \text{frame\$V2}$
Parent Izq	($(\text{frame\$V1})$
Parent Der)	$(\text{frame\$V1})$
Potencia	^	$\text{frame\$V1} \wedge \text{frame\$V2}$

- RAÍCES Y EXPONENCIAL -

Raíz Cuadrada	\sqrt{x}	$\text{sqrt}(x)$
Exponencial	e^x	$\text{exp}(x)$

- LOGARITMOS -

Logaritmo	$\log(x)$	$\text{log}(x)$
Logaritmo Base 2	$\log_2(x)$	$\text{log2}(x)$
Logaritmo Base 10	$\log_{10}(x)$	$\text{log10}(x)$
Logaritmo Base n	$\log_n(x)$	$\text{log}(x,n)$

- FACTORIAL -

Beta	$beta(a, b)$	<code>beta(a, b)</code>
Lbeta	$lbeta(a, b)$	<code>lbeta(a, b)</code>
Gamma	$gamma(x)$	<code>gamma(x)</code>
Lgamma	$lgamma(x)$	<code>lgamma(x)</code>
Choose	$choose(n, k)$	<code>choose(n, k)</code>
Lchoose	$lchoose(n, k)$	<code>lchoose(n, k)</code>
Factorial	$factorial(x)$	<code>factorial(x)</code>
Lfactorial	$lfactorial(x)$	<code>lfactorial(x)</code>

- TRIGONOMETRÍA -

Coseno	$\cos(x)$	<code>cos(x)</code>
Seno	$\sin(x)$	<code>sin(x)</code>
Tangente	$\tan(x)$	<code>tan(x)</code>
Arcocoseno	$\arccos(x)$	<code>acos(x)</code>
Arcoseno	$\arcsin(x)$	<code>asin(x)</code>
Arcotangente	$\arctan(x)$	<code>atan(x)</code>

- SIMULADORES DE RUIDO -

Unif(0,1)	$r_{unif}(n)$	<code>runif(n)</code>
Exp(mean)	$r_{exp}(n, mean)$	<code>rexp(n, mean)</code>
N(0,1)	$r_{norm}(n)$	<code>rnorm(n)</code>
St(df,0,1)	$r_{t}(n, df)$	<code>rt(n, df)</code>
Cauchy(0,1)	$r_{cauchy}(n)$	<code>rcauchy(n)</code>

1.2 Random Simulation

En este apartado vamos a especificar las distintas distribuciones que vamos a permitir ofrecer para generar las simulaciones. Las dividiremos en continuas y discretas.

- DISTRIBUCIONES CONTINUAS -

1. Uniform	<code>runif</code>	10. Gamma	<code>rgamma</code>
2. Exponential	<code>rexp</code>	11. Inverse Gamma	<code>rinv.gamma(NSTR)</code>
3. Normal	<code>rnorm</code>	12. Gamma-Gamma	<code>rgamma.gamma(NSTR)</code>
4. Cauchy	<code>rcauchy</code>	13. Weibull	<code>rweibull</code>
5. Standard Student-t	<code>rt</code>	14. Pareto	<code>rpareto1(actuar)</code>
6. Student-t	<code>NSTR</code>	15. Inverse Pareto	<code>rinv.pareto1(NSTR)</code>
7. Chi-squared	<code>rchisq(nlf=0)</code>	16. Log-normal	<code>rlnorm</code>
8. Non-central Chi-squared	<code>rchisq</code>	17. Beta	<code>rbeta</code>
9. Snedecor F	<code>rf</code>	18. Dirichlet	<code>rdirichlet(LearnBayes)</code>

- DISTRIBUCIONES DISCRETAS -

1. Bernoulli	<code>rbinom(size=1)</code>
2. Binomial	<code>rbinom</code>
3. Poisson	<code>rpois</code>
4. Geometric	<code>rgeom</code>
5. Negative-Binomial	<code>simnb(degreet)</code>
6. Binomial-Beta	<code>rbinom.beta(NSTR)</code>
7. Poisson-Gamma	<code>rpois.gamma(NSTR)</code>
8. Negative Binomial-Beta	<code>rnbinom.beta(NSTR)</code>
9. Multinomial	<code>rmultinom</code>

1.2.1 Distribuciones Continuas

Uniform distribution

Parámetros:

	Default
a (<i>Min</i>)	0
b (<i>Max</i>)	1

Comando R:

```
runif (nsim, a, b)
```

Librería Java:

```
public double[] uniform ( int nsim, double a, double b )
```

Notation	$x \sim U(a, b), \quad a \leq x \leq b$
Parameters	$b > a$
Density function	$f(x) = \frac{1}{b-a}, \theta \in [a, b]$
Mean	$E(x) = \frac{a+b}{2}$
Variance	$var(x) = \frac{(b-a)^2}{12}$

Exponential distribution

Parámetros:

	Default
β (Rate)	\emptyset

Comando R:

```
rexp (nsim, Rate)
```

Librería Java:

```
public double[] exponential ( int nsim, double beta )
```

Notation	$x \sim Expon(\beta),$	$-\infty < x < \infty$
Parameters	$\beta > 0$	
Density function	$f(x) = \beta e^{-\beta x}$	
Mean	$E(x) = \frac{1}{\beta}$	
Variance	$var(x) = \frac{1}{\beta^2}$	

Normal distribution

Parámetros:

	Default
μ (Mean)	0
σ (Sd)	1

Comando R:

```
rnorm (n, mean = 0, sd = 1)
```

Librería Java:

```
public double[] normal ( int nsim, double mu, double sigma )
```

Notation	$x \sim N(\mu, \sigma),$	$-\infty < x < \infty$
Parameters	$-\infty < \mu < \infty$ location;	$\sigma > 0$ scale
Density function	$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}}$	$\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$
Mean	$E(x) = \mu$	
Variance	$var(x) = \sigma^2$	
Default values	$\mu = 0; \sigma = 1$	

Cauchy distribution

Parámetros:

	Default
μ (Location)	0
σ (Scale)	1

Comando R:

```
rcauchy (n, location = 0, scale = 1)
```

Librería Java:

```
public double[] cauchy ( int nsim, double mu, double sigma )
```

Notation	$x \sim Ca_{\mu,\sigma}, \quad -\infty < x < \infty$
Parameters	$-\infty < \mu < \infty$ location; $\sigma > 0$ scale
Density function	$f(x) = [\pi\sigma \left(1 + \left(\frac{x-\mu}{\sigma}\right)^2\right)]^{-1}$
Mean	undefined
Variance	undefined

Standard Student-t distribution**Parámetros:**

	<i>Default</i>
ν (Df)	\emptyset

Comando R:

```
rt (n, df)
```

Librería Java:

```
public double[] standarStudentT ( int nsim, double nu )
```

Notation	$x \sim St_{\nu}, \quad -\infty < x < \infty$
Parameters	$\nu > 0$ degrees of freedom
Density function	$f(x) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi}} \left[1 + \frac{x^2}{\nu}\right]^{-(\nu+1)/2}$
Mean	$E(x) = 0$, for $\nu > 1$
Variance	$var(x) = \frac{\nu}{\nu-2}$, for $\nu > 2$

Student-t distribution**Parámetros:**

	<i>Default</i>
ν (Df)	\emptyset
μ (Location)	0
σ (Scale)	1

Comando R:

```
(rt(n,df)+location)*scale
```

Librería Java:

```
public double[] studentT ( int nsim, double nu, double mu, double sigma)
```

Notation	$x \sim St(\mu, \sigma, \nu), \quad -\infty < x < \infty$
Parameters	$\nu > 0$ degrees of freedom, $-\infty < \mu < \infty$ location; $\sigma > 0$ scale
Density function	$f(x) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2) \sqrt{\nu\pi}\sigma^2} \left[1 + \frac{1}{\nu} \left(\frac{x-\mu}{\sigma}\right)^2\right]^{-(\nu+1)/2}$
Mean	$E(x) = \mu, \text{ for } \nu > 1$
Variance	$var(x) = \frac{\nu}{\nu-2} \sigma^2, \text{ for } \nu > 2$

Chi-squared distribution**Parámetros:**

	<i>Default</i>
ν (Df)	\emptyset

Comando R:

```
rchisq (n, df, ncp=0)
```

Librería Java:

```
public double[] chiSquare (int nsim, double nu )
```

Notation	$x \sim \chi_\nu^2, \quad x > 0$
Parameters	$\nu > 0$ degrees of freedom
Density function	$f(x) = \left(\frac{x}{2}\right)^{\nu/2} \frac{\Gamma(\nu/2)}{x} e^{-x/2}$
Mean	$E(x) = \nu$
Variance	$var(x) = 2\nu$

Non-central Chi-squared distribution**Parámetros:**

	<i>Default</i>
ν (Df)	\emptyset
λ (Ncp)	\emptyset

Comando R:

```
rchisq (n, df, ncp)
```

Librería Java:

```
public double[] nonCentralChiSquare (int nsim, double nu, double lambda )
```

Notation	$x \sim \chi^2(\nu, \lambda), \quad x > 0$
Parameters	$\nu > 0$ degrees of freedom; $\lambda > 0$ non-centrality parameter
Density function	$f(x) = e^{-\lambda/2} \sum_{r=0}^{\infty} \frac{(\lambda/2)^r}{r!} f_{\chi_{\nu+2r}^2}(x)$ with $f_{\chi_{\nu}^2}(x)$ the chi-squared density function on x with ν d.f.
Mean	$E(x) = \nu + \lambda$
Variance	$var(x) = 2(\nu + 2\lambda)$

Snedecor F distribution**Parámetros:**

	<i>Default</i>
α (Df1)	\emptyset
β (Df2)	\emptyset

Comando R:

```
rf (n, df1, df2)
```

Librería Java:

```
public double[] snedecorF ( int nsim, double alpha, double beta )
```

Notation	$x \sim F_{\alpha, \beta}, \quad x > 0$
Parameters	$\alpha > 0, \beta > 0$ degrees of freedom
Density function	$f(x) = \frac{\Gamma((\alpha+\beta)/2)\alpha^{\alpha/2}\beta^{\beta/2}}{\Gamma(\alpha/2)\Gamma(\beta/2)} \frac{x^{\alpha/2-1}}{(\beta+\alpha x)^{(\alpha+\beta)/2}}$
Mean	$E(x) = \frac{\beta}{\beta-2},$ for $\beta > 2$
Variance	$var(x) = \frac{2\beta^2(\alpha+\beta-2)}{\alpha(\beta-2)^2(\beta-4)},$ for $\beta > 4$

Gamma distribution**Parámetros:**

	<i>Default</i>
α (Shape)	\emptyset
β (Rate)	\emptyset

Comando R:

```
rgamma (n, shape, rate = 1)
```

Librería Java:

```
public double[] gamma ( int nsim, double alpha, double beta)
```

Notation	$x \sim Ga(\alpha, \beta), \quad x > 0$
Parameters	$\alpha > 0, \quad \beta > 0$
Density function	$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x}$
Mean	$E(x) = \frac{\alpha}{\beta}$
Variance	$var(x) = \frac{\alpha}{\beta^2}$

Inverse-Gamma distribution.**Parámetros:**

	<i>Default</i>
α (Shape)	\emptyset
β (Scale)	\emptyset

Comando R:

```
rinv.gamma=function(nsim,shape=alpha,rate=beta){
1/rgamma(nsim,alpha,beta)
}
```

Librería Java:

```
public double[] gammaInverse ( int nsim, double alpha, double beta )
```

Notation	$x \sim \text{Inv} - \text{Gamma}(\alpha, \beta), \quad x > 0$
Parameters	$\alpha > 0, \quad \beta > 0$
Density function	$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{-(\alpha+1)} e^{-\beta/x}$
Mean	$E(x) = \frac{\beta}{\alpha-1}, \quad \text{for } \alpha > 1$
Variance	$\text{var}(x) = \frac{\beta^2}{(\alpha-1)^2(\alpha-2)}, \quad \text{for } \alpha > 2$

Gamma-Gamma distribution

Parámetros:

	Default
α	\emptyset
β	\emptyset
ν	\emptyset

Comando R:

```
rgamma.gamma=function(nsim,alpha,beta,nu){
theta=rgamma(nsim,alpha,beta)
x=vector(length=nsim)
for(i in 1:nsim){
theta[i]=rgamma(1,nu,theta[i])
}
return(theta)
}
```

Librería Java:

```
public double[] gammagamma ( double nsim, double alpha, double beta,
double nu)
```

Notation	$x \sim \text{Gamma} - \text{Gamma}(\alpha, \beta, \nu), \quad x > 0$
Parameters	$\alpha > 0, \quad \beta > 0, \quad \nu > 0$
Density function	$f(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{\Gamma(\alpha+\nu)}{\Gamma(\nu)} \frac{x^{\nu-1}}{(\beta+x)^{\alpha+\nu}}$
Mean	$E(x) = \nu \frac{\beta}{\alpha-1}, \quad \text{for } \alpha > 1$
Variance	$\text{var}(x) = \frac{\beta^2(\nu^2+\nu(\alpha-1))}{(\alpha-1)^2(\alpha-2)}, \quad \text{for } \alpha > 2$

Weibull distribution

Parámetros:

	Default
α (Shape)	\emptyset
β (Scale)	1

Comando R:

```
rweibull (n, shape, scale = 1)
```

Librería Java:

```
public double[] weibull ( int nsim, double alpha, double beta )
```

Notation	$x \sim Weib(\alpha, \beta), \quad x \geq 0$
Parameters	$\alpha > 0, \quad \beta > 0$
Density function	$f(x) = \frac{\alpha}{\beta} \left(\frac{x}{\beta}\right)^{\alpha-1} e^{-(x/\beta)^\alpha}$
Mean	$E(x) = \beta \Gamma(1 + 1/\alpha), \quad \text{for } \alpha > 1$
Variance	$var(x) = \beta^2 \Gamma(1 + 2/\alpha) - \Gamma(1 + 1/\alpha)^2, \quad \text{for } \alpha > 2$

Pareto

Parámetros:

	Default
α (shape)	\emptyset
β (min)	\emptyset

Comando R:

```
library(actuar)
rpareto1 (nsim, shape, min)
```

Librería Java:

```
public double[] pareto (int nsim, double alpha, double beta )
```

Notation	$x \sim Pa(\alpha, \beta), \quad x > \beta$
Parameters	$\alpha < x$ location; $\beta > 0$ minimum
Density function	$f(x) = \frac{\beta \alpha^\beta}{x^{\beta+1}}$
Mean	$E(x) = \frac{\alpha \beta}{\beta-1}, \quad \text{for } \beta > 1$
Variance	$var(x) = \frac{\alpha^2 \beta}{(\beta-1)^2(\beta-2)}, \quad \text{for } \beta > 2$

Inverse Pareto distribution

Parámetros:

	<i>Default</i>
α (shape)	\emptyset
β (mininv)	\emptyset

Comando R:

`library(VGAM)`

```
library(actuar)
rinv.pareto1=function(nsim,shape=alpha,mininv=beta){
1/rpareto1(nsim,alpha,beta)
}
```

Librería Java:

```
public double[] inversePareto ( double nsim, double alpha, double beta )
```

Notation	$x \sim Pa(\alpha, \beta), \quad 0 < x < 1/\beta$
Parameters	$\alpha > 0$ location; $\beta > 0$ mininv
Density function	$f(x) = \frac{\beta\alpha^\beta}{x^{\beta+1}}$
Mean	$E(x) = \frac{\beta}{\alpha(\beta+1)}$
Variance	$var(x) = \frac{\beta}{\alpha^2(\beta+1)^2(\beta+2)}$

Log-normal distribution

Parámetros:

	<i>Default</i>
μ (Meanlog)	0
σ (Sdlog)	1

Comando R:

`rlnorm (n, meanlog = 0, sdlog = 1)`

Librería Java:

```
public double[] logNormal ( int nsim, double mu, double sigma )
```

Notation	$x \sim \text{LogN}(\mu, \sigma), \quad x > 0$
Parameters	$-\infty < \mu < \infty$ location; $\sigma > 0$ scale
Density function	$f(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(\log(x)-\mu)^2}{2\sigma^2}\right)$
Mean	$E(x) = e^{\mu+\sigma^2/2}$
Variance	$\text{var}(x) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)$

Beta distribution

Parámetros:

	Default
α (Shape1)	\emptyset
β (Shape2)	\emptyset

Comando R:

```
rbeta (n, shape1, shape2, ncp = 0)
```

Librería Java:

```
public double[] beta ( int nsim, double alpha, double beta )
```

Notation	$x \sim \text{Beta}(\alpha, \beta), \quad x \in [0, 1]$
Parameters	$\alpha > 0, \beta > 0$, shapes
Density function	$f(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1}$
Mean	$E(x) = \frac{\alpha}{\alpha+\beta}$
Variance	$\text{var}(x) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Dirichlet distribution

Parámetros:

	Default	Aclaración
Alpha (Es un vector)	\emptyset	Es un vector.

Comando R:

```
library(LearnBayes)
rdirichlet (nsim, par=( $\alpha_1, \dots, \alpha_{k+1}$ ))
```

Ojo porque el resultado son tres elementos. Los representaremos como tres columnas de nuestro frame: NombreColumna_1, NombreColumna_2 y NombreColumna_3.

Librería Java:

```
public double[][] dirichlet ( int n, double[] alpha )
```

Notation	$x \sim \text{Dirichlet}(\alpha_1, \dots, \alpha_{k+1}), \quad x = (x_1, \dots, x_k)$ $0 < x_i < 1, \quad \sum_{i=1}^k x_i \leq 1$
Parameters	$\alpha_i > 0$
Density function	$f(x) = \frac{\Gamma(\sum_{i=1}^{k+1} \alpha_i)}{\prod_{i=1}^{k+1} \Gamma(\alpha_i)} \left(1 - \sum_{i=1}^k x_i\right)^{\alpha_{k+1}-1} \prod_{i=1}^k x_i^{\alpha_i-1}$
Mean	$E(x_i) = \frac{\alpha_i}{\sum_{l=1}^{k+1} \alpha_l}$
Variance	$var(x_j) = \frac{E(x_i)[1-E(x_i)]}{1+\sum_{l=1}^{k+1} \alpha_l}$
Covariance	$C(x_i, x_j) = \frac{-E(x_i)E(x_j)}{1+\sum_{l=1}^{k+1} \alpha_l}$

1.2.2 Distribuciones Discretas

Bernoulli distribution

Parámetros:

	<i>Default</i>
p (Prob)	\emptyset

Comando R:

```
rbinom (n, size = 1, prob)
```

Librería Java:

```
public double[] bernoulli ( int nsim, double p )
```

Notation	$x \sim Br(p), \quad x = 0, 1$
Parameters	$0 < p < 1$
Probability function	$p(x) = p^x(1-p)^{1-x}$
Mean	$E(x) = p$
Variance	$var(x) = p(1-p)$

Binomial distribution

Parámetros:

	Default
n (Size)	\emptyset
p (Prob)	\emptyset

Comando R:

```
rbinom (n, size, prob)
```

Librería Java:

```
public double[] binomial ( int nsim, double n, double p )
```

Notation	$x \sim Bin(n, p), \quad x = 0, 1, 2, \dots, n$
Parameters	$n = 1, 2, \dots; 0 < p < 1$
Probability function	$p(x) = \binom{n}{x} p^x (1-p)^{n-x}$
Mean	$E(x) = np$
Variance	$var(x) = np(1-p)$

Poisson distribution

Parámetros:

	Default
Lambda (λ)	\emptyset

Comando R:

```
rpois (n, lambda)
```

Librería Java:

```
public double[] poisson ( int n, double lambda )
```

Notation	$x \sim Po(\lambda), \quad x = 0, 1, 2, \dots$
Parameters	$\lambda > 0$
Probability function	$p(x) = \frac{\lambda^x}{x!} e^{-\lambda}$
Mean	$E(x) = \lambda$
Variance	$var(x) = \lambda$

Geometric distribution.

Parámetros:

	<i>Default</i>
p (Prob)	\emptyset

Comando R:

```
library(degreenet) simnb(nsim,v=c(1,p))
```

Librería Java:

```
public double[] geometric (int nsim, double p )
```

Notation	$x \sim Geo(p)$
Parameters	$0 < p \leq 1$
Probability function	$p(x) = p(1 - p)^x \quad x = 0, 1, 2, \dots$
Mean	$E(x) = p$
Variance	$var(x) = \frac{1-p}{p^2}$

Negative-Binomial distribution

Parámetros:

	<i>Default</i>
r	\emptyset
p	\emptyset

Comando R:

```
library(degreenet)
simnb(nsim,v=c(r,p))
```

Librería Java:

```
public double[] binomialNegative ( int nsim, double r, double p )
```

Notation	$x \sim NBin(p, r)$
Parameters	$0 < p < 1; r = 1, 2, \dots$
Probability function	$p(x) = \binom{r+x-1}{r-1} p^r (1-p)^x$
Mean	$E(x) = rp$
Variance	$var(x) = r \frac{1-p}{p^2}$

Binomial-Beta distribution**Parámetros:**

	Default
n	\emptyset
α	\emptyset
β	\emptyset

Comando R:

```
rbinom.beta=function(nsim,n,alpha,beta){
p=rbeta(nsim,alpha,beta)
x=vector(length=nsim)
for(i in 1:nsim){
x[i]=rbinom(1,n,p[i])
}
return(x)}
```

Librería Java:

```
public double[] betaBinomial ( int nsim, double n, double alpha, double
beta )
```

Notation	$x \sim BinBe(n, \alpha, \beta), \quad x = 0, 1, \dots, n$
Parameters	$n = 1, 2, \dots; \alpha > 0, \beta > 0$
Probability function	$p(x) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \binom{n}{x} \Gamma(\alpha+x)\Gamma(\beta+n-x)$
Mean	$E(x) = n \frac{\alpha}{\alpha+\beta}$
Variance	$var(x) = n \frac{\alpha\beta(\alpha+\beta+n)}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Poisson-Gamma distribution

Parámetros:

	<i>Default</i>
α	\emptyset
β	\emptyset
λ	\emptyset

Comando R:

```
library(degreenet)

rpois.gamma=function(nsim,alpha,beta,lambda){
theta=rgamma(nsim,alpha,beta)
x=vector(length=nsim)
for(i in 1:nsim){
x[i]=rpois(1,theta[i])
}
return(x)
}
```

Librería Java:

```
public double[] poissonGamma ( int nsim, double alpha, double beta,
double lambda )
```

Notation	$x \sim PoGa(\alpha, \beta, \lambda), \quad x = 0, 1, 2, \dots$
Parameters	$\lambda > 0, \alpha > 0, \beta > 0$
Probability function	$p(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} \frac{\Gamma(\alpha+x)}{x!} \frac{\lambda^x}{(\beta+\lambda)^{\alpha+x}}$
Mean	$E(x) = \lambda \frac{\alpha}{\beta}$
Variance	$var(x) = \frac{\lambda\alpha}{\beta} \left(1 + \frac{\lambda}{\beta} \right)$

Negative Binomial-Beta distribution

Parámetros:

	<i>Default</i>
α (Shape1)	\emptyset
β (Shape2)	\emptyset
r	\emptyset

Comando R:

```
library(degreenet);

rnbinom.beta=function(nsim,alpha,beta,r){
p=rbeta(nsim,alpha,beta)
x=vector(length=nsim)
for(i in 1:nsim){
x[i]=simnb(1,v=c(r,p[i]))}
return(x)
}
```

Librería Java:

```
public double[] betaBinomialNegative ( int nsim, double alpha, double
beta, double r )
```

Notation	$x \sim NBinBe(\alpha, \beta, r)$
Parameters	$\alpha > 0, \beta > 0, r = 1, 2, \dots$
Probability function	$p(x) = \binom{r+x-1}{r-1} \frac{\Gamma(\alpha+\beta)\Gamma(\alpha+r)}{\Gamma(\alpha)\Gamma(\beta)} \frac{\Gamma(\beta+x)}{\Gamma(\alpha+\beta+r+x)}$
Mean	$E(x) = \frac{r\beta}{\alpha-1}$
Variance	$var(x) = \frac{r\beta}{\alpha-1} \left[\frac{\alpha+\beta+r-1}{\alpha-2} + \frac{r\beta}{(\alpha-1)(\alpha-2)} \right]$

Multinomial distribution.**Parámetros:**

	<i>Default</i>	Aclaración
n (Size)	\emptyset	
p (Prob)	\emptyset	Vector de probabilidades, comprobar que suman uno.

Comando R:

```
rmultinom(nsim, size = n, prob = c(p1, ...pk))
```

Librería Java:

```
public double[][] multinomial ( int nsim, double n, double[] p )
```

Notation	$x \sim Mn(n; p_1, \dots, p_k), \quad \sum_{i=1}^k x_i \leq n, \quad x_i = 0, 1, 2, \dots$
Parameters	$n = 1, 2, \dots, p_j \in (0, 1), \quad \sum_{i=1}^k p_j \leq 1$
Probability function	$p(x) = \frac{n!}{\prod_{i=1}^{k+1} x_i!} \prod_{i=1}^{k+1} p_i^{x_i}$ with $p_{k+1} = 1 - \sum_{i=1}^k p_i; \quad x_{k+1} = n - \sum_{i=1}^k x_i$
Mean	$E(x_i) = np_j$
Variance	$var(x_i) = np_i(1 - p_i)$
Covariance	$C(x_i, x_j) = -np_i p_j$

1.3 Modelos Bayesianos

1.3.1 Uniform Model

Let x_1, \dots, x_n iid Uniform $x_i | \theta \sim Un(0, \theta) \quad \theta > 0 \quad i = 1, \dots, n$. Let $m_n = \max\{x_1, x_2, \dots, x_n\}$

Objective Analysis

Prior	Posterior	Posterior Predictive
$\frac{1}{\theta}$	$Pa(\theta n, m_n)$	$\frac{n}{n+1} Un(x 0, m_n), \quad \text{if } x \leq m_n$ $\frac{1}{n+1} Pa(x n, m_n), \quad \text{if } x > m_n$

Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$Pa(\theta \alpha, \beta)$	$Pa(\theta \alpha + n, \beta_n)$	$\frac{\alpha}{\alpha+1} Un(x 0, \beta), \quad \text{if } x \leq \beta$ $\frac{1}{\alpha+1} Pa(x \alpha, \beta), \quad \text{if } x > \beta$	$\frac{\alpha+n}{\alpha+n+1} Un(x 0, \beta_n), \quad \text{if } x \leq \beta_n$ $\frac{1}{\alpha+n+1} Pa(x \alpha + n, \beta_n), \quad \text{if } x > \beta_n$

with known α and β and $\beta_n = \max\{\beta, m_n\}$.

1.3.2 Bernoulli model

Let x_1, \dots, x_n iid Bernoulli, $x_i \sim B(p)$ $i = 1, \dots, n$. Let $r = \sum_{i=1}^n x_i$. Let $x_0 \sim B(p)$ a new observation to predict.

Objective Analysis

Prior		Posterior	Posterior Predictive
Uniform	1	$Be(p r + 1, n - r + 1)$	$BeBin(x_0 1 + r, 1 + n - r, 1)$
Jeffreys	$\frac{1}{\pi}p^{-1/2}(1 - p)^{-1/2}$	$Be(p \frac{1}{2} + r, \frac{1}{2} + n - r)$	$BeBin(x_0 \frac{1}{2} + r, \frac{1}{2} + n - r, 1)$
Reference			
Novick and Halls ¹	$p^{-1}(1 - p)^{-1}$	$Be(p r, n - r)$	$BeBin(x_0 r, n - r, 1)$

¹ See Novick and Hall (1965); this prior is Uniform on $\theta = \log \frac{p}{(1-p)}$.

Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$Be(p \alpha, \beta)$	$Be(p \alpha + r, \beta + n - r)$	$BeBin(x_0 \alpha, \beta, 1)$	$BeBin(x_0 \alpha + r, \beta + n - r, 1)$

with known α and β .

1.3.3 Binomial model

Let x_1, \dots, x_n independent Binomial, $x_i \sim Bin(n_i, p)$ $i = 1, \dots, n$. Let $r = \sum_{i=1}^n x_i$ and $m = \sum_{i=1}^n n_i$. Let $x_0 \sim Bin(n_0, p)$ a new observation to predict. With known n_0 .

Objective Analysis

Prior		Posterior	Posterior Predictive
Uniform	1	$Be(p r+1, m-r+1)$	$BeBin(x_0 1+r, 1+m-r, 1)$
Jeffreys	$\frac{1}{\pi}p^{-1/2}(1-p)^{-1/2}$	$Be(p \frac{1}{2}+r, \frac{1}{2}+m-r)$	$BeBin(x_0 \frac{1}{2}+r, \frac{1}{2}+m-r, n_0)$
Reference			
Novick and Halls ¹	$p^{-1}(1-p)^{-1}$	$Be(p r, m-r)$	$BeBin(x_0 r, m-r, n_0)$

¹ See Novick and Hall (1965); this prior is Uniform on $\theta = \log \frac{p}{(1-p)}$.

Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$Be(p \alpha, \beta)$	$Be(p \alpha+r, \beta+m-r)$	$BeBin(x_0 \alpha, \beta, n_0)$	$BeBin(x_0 \alpha+r, \beta+m-r, n_0)$

with known α and β .

1.3.4 Poisson Model

Let x_1, \dots, x_n iid Poisson, $x_i|\lambda \sim Po(\lambda)$ $i = 1, \dots, n$. Let $r = \sum_{i=1}^n x_i$. Let $x_0 \sim Po(\lambda)$ a new observation to predict.

Objective Analysis

Prior		Posterior	Posterior Predictive
Uniform	1	$Ga(\lambda r+1, n)$	$Pg(x_0 1+r, n, 1)$
Jeffreys	$\lambda^{-1/2}$	$Ga(\lambda r+1/2, n)$	$Pg(x_0 r+1/2, n, 1)$
Reference			

Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$Ga(\lambda \alpha, \beta)$	$Ga(\lambda \alpha + r, \beta + n)$	$Pg(x_0 \alpha, \beta, 1)$	$Pg(x_0 \alpha + r, \beta + n, 1)$

with known α and β .

1.3.5 Exponential model

Let x_1, \dots, x_n iid Exponential, $x_i|\lambda \sim Exp(\lambda)$ where $\lambda > 0$, $x_i > 0$, $i = 1, \dots, n$. Let $r = \sum_{i=1}^n x_i$. Let $x_0 \sim Exp(\lambda)$ a new observation to predict.

Objective Analysis

Prior		Posterior	Posterior Predictive
Jeffreys	λ^{-1}	$Ga(\lambda n, r)$	$Gg(x_0 n, r, 1)$

Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$Ga(\lambda \alpha, \beta)$	$Ga(\lambda n + \alpha, \beta + r)$	$Gg(x_0 \alpha, \beta, 1)$	$Gg(x_0 n + \alpha, \beta + r, 1)$

with known α and β .

1.3.6 Normal Model

Let x_1, \dots, x_n iid Normal ($x_i|\mu, \sigma^2$) $\sim N(\mu, \sigma^2)$ $i = 1, \dots, n$. Let $\bar{x} = \sum_{i=1}^n x_i/n$. Let $x_0 \sim N(\mu, \sigma^2)$ a new observation to predict.

σ^2 known

1. Objective Analysis

Prior	Posterior	Posterior Predictive
1	$N(\mu \bar{x}, \sigma^2/n)$	$N(x_0 \bar{x}, \sigma^2(1 + 1/n))$

2. Subjective Analysis

Prior	Posterior	Prior Predictive	Posterior Predictive
$N(\mu \mu_0, \tau_0^2)$	$N(\mu \mu_n, \tau_n^2)$	$N(x_0 \mu_0, \sigma^2 + \tau_0^2)$	$N(x_0 \mu_n, \sigma^2 + \tau_n^2)$

with known μ_0 and *vir*

$$\kappa_n = \sigma^2 / (n\tau_0^2 + \sigma^2)$$

$$\mu_n = \mu_0\kappa_n + \bar{x}(1 - \kappa_n)$$

$$\tau_n^2 = \tau_0^2\kappa_n.$$

 μ known

Let $t = \sum_i (x_i - \mu)^2$.

1. Objective Analysis

Prior		Posterior	Posterior Predictive
Uniform	1	$IGa(\sigma^2 \frac{n}{2} - 1, \frac{t}{2})$	$St(x_0 \mu, \frac{t}{n-2}, n-2)$
Jeffreys	$1/\sigma^2$	$IGa(\sigma^2 \frac{n}{2}, \frac{t}{2})$	$St(x_0 \mu, \frac{t}{n}, n)$
Reference			
MDIP			

2. Subjective Analysis: general parameterization

Prior	Posterior
$IGa(\sigma^2 \alpha, \beta)$	$IGa(\sigma^2 \alpha + n/2, \beta + t/2)$
Prior Predictive	Posterior Predictive
$St(x_0 \mu, \beta/\alpha, 2\alpha)$	$St(x_0 \mu, \frac{\beta+t/2}{\alpha+n/2}, 2\alpha + n)$

3. Subjective Analysis: another parameterization

Prior	Posterior
$IGa(\sigma^2 \nu_0 + 1, \nu_0\sigma_0^2)$	$IGa(\sigma^2 \nu_0 + 1 + n/2, \nu_0\sigma_0^2 + t/2)$
Prior Predictive	Posterior Predictive
$St(x_0 \mu, \frac{\nu_0\sigma_0^2}{\nu_0+1}, 2(\nu_0 + 1))$	$St(x_0 \mu, \frac{\nu_0\sigma_0^2+t/2}{\nu_0+1+n/2}, 2(\nu_0 + 1) + n)$

with ν_0 y σ_0^2 known.

μ and σ^2 unknown

Let $s^2 = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2$

1. Objective Analysis

Prior		(marginal) Posterior	Posterior Predictive
Uniform	1	$St(\mu \bar{x}, \frac{(n-1)s^2}{n(n-3)}, n - 3)$ $GaI(\sigma^2 \frac{n-3}{2}, \frac{(n-1)s^2}{2})$	$St(x_0 \bar{x}, (\frac{1}{n} + 1)\frac{(n-1)s^2}{n-3}, n - 3)$
Jeffreys	$1/\sigma^4$	$St(\mu \bar{x}, \frac{(n-1)s^2}{n(n+1)}, n + 1)$ $GaI(\sigma^2 \frac{n+1}{2}, \frac{(n-1)s^2}{2})$	$St(x_0 \bar{x}, (\frac{1}{n} + 1)\frac{(n-1)s^2}{n+1}, n + 1)$
Reference MDIP	$1/\sigma^2$	$St(\mu \bar{x}, \frac{s^2}{n}, n - 1)$ $GaI(\sigma^2 \frac{n-1}{2}, \frac{(n-1)s^2}{2})$	$St(x_0 \bar{x}, (\frac{1}{n} + 1)s^2, n - 1)$

2. Subjective Analysis

Prior	(marginal) Posterior
$St(\mu \mu_0, \frac{\nu_0}{\nu_0+1}\sigma_0^2, 2(\nu_0+1))$	$St(\mu \mu_n, \frac{\sigma_n^2}{\kappa_n}, \nu_n)$
$GaI(\sigma^2 \nu_0+1, \nu_0\sigma_0^2)$	$GaI(\sigma^2 \frac{\nu_n}{2}, \frac{\nu_n\sigma_n^2}{2})$
Prior Predictive	Posterior Predictive
$St(x_0 \mu_0, \sigma_0^2(1 + \frac{1}{\kappa_0}), 2\nu_0+2)$	$St(x_0 \mu_n, \sigma_n^2(1 + \frac{1}{\kappa_n}), \nu_n)$

with ν_0, κ_0, ν_0 and σ_0^2 known, and

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

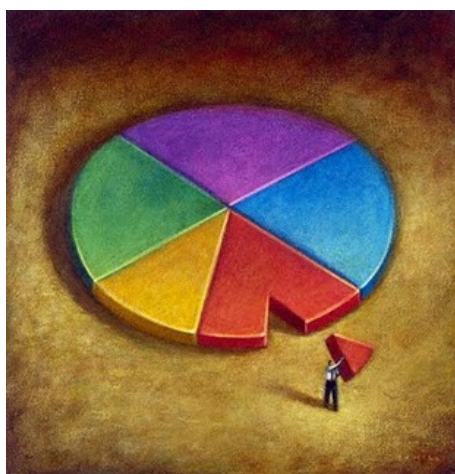
$$\mu_n = \frac{\mu_0 \kappa_0 + n \bar{x}}{\kappa_0 + n}$$

$$\kappa_n = \kappa_0 + n$$

$$\nu_n = 2(\nu_0 + 1) + n$$

$$s^2 = \frac{1}{n-1} \sum_i (x_i - \bar{x})^2$$

$$\nu_n \sigma_n^2 = 2\nu_0 \sigma_0^2 + (n-1)s^2 + \frac{\kappa_0 n}{\kappa_0 + n} (\bar{x} - \mu_0)^2$$



UNIVERSIDAD DE MURCIA

ANEXO II

Aplicación OBANSoft

Resumen del póster presentado en el congreso
“Valencia International Meeting on Bayesian
Statistics, 2010 ISBA World Meeting”. Junio 2010

Ninth Valencia International Meeting on Bayesian Statistics

2010 ISBA World Meeting

Co-sponsored by the *Universitat de València* (Spain), and by the *International Society for Bayesian Analysis* (ISBA), the *9th Valencia International Meeting on Bayesian Statistics* and the *2010 World Meeting of the International Society for Bayesian Analysis* are being jointly held in Benidorm (Alicante, Spain) from Thursday June 3rd to Tuesday June 8th, both inclusive. This is preceded by a half-day set of postgraduate tutorials on the morning of Thursday June 3rd.

The scientific programme includes 24 lectures invited by the Valencia 9 Scientific Committee, whose discussion will be initiated by an invited discussant, 40 plenary talks selected through a blinded review process by the ISBA 2010 Program Committee, and about 350 contributed papers to be presented in poster form in 5 evening plenary sessions. This booklet contains the complete joint programme, the abstracts of all contributions (in alphabetical order by first named author), the list of participants, and an abbreviated version of the Bayesian singalong book.

Local Organizer

José M. Bernardo (*Universitat de València, Spain*)

Valencia 9 Scientific Committee

Susie Bayarri (*Universitat de València, Spain*)

James O. Berger (*Duke University, USA*)

José M. Bernardo (*Universitat de València, Spain*)

A. Philip Dawid (*University of Cambridge, UK*)

David Heckerman (*Microsoft Research, USA*)

Adrian F. M. Smith (*Director General of Science and Research, UK*)

Mike West (*Duke University, USA*)

ISBA Programme Committee

Cathy Chen (*Feng Chia University, Taiwan*)

Andrés Christen (*CIMAT, México*)

Simon Godsill (*University of Cambridge, UK*)

Aparna Huzurbazar (*Los Alamos National Laboratory, USA*)

Herbie Lee (*University of California, Santa Cruz, USA*)

Xiao-Li Meng (*Harvard University, USA*)

Kerrie Mengersen (*Queensland University of Technology, Australia*)

Peter Müller (*M. D. Anderson Cancer Center, USA*)

Sonia Petrone (*Università Bocconi, Italy*)

Gareth Roberts (*University of Warwick, UK*)

Alexandra Schmidt (*Universidade Federal do Rio de Janeiro, Brazil*)

Social Programme

The opening ceremony will take place in the Auditorium of Gran Hotel Bali, Benidorm, at 09h15 of Friday June 4th, 2010. The ISBA General Body Meeting will be on afternoon of Sunday June 6th. All members are invited to attend for discussion of ISBA business including planning and decisions for future World Meetings. The conference banquet will be held on the evening of Tuesday June 8th. This will be followed by the traditional Valencia meeting cabaret. The seaside location of the venue provides an ideal leisure setting for accompanying persons during the conference working hours.

Proceedings

Bayesian Statistics 9, the Valencia 9 proceedings, will be published by Oxford University Press. This will include the invited papers and their discussions. By June 28th 2010, the invited discussants and all delegates wishing to participate in the discussion should send their written discussions to the author(s) of the invited papers, and to the local organiser. Contributions should not exceed *six* typeset pages (including figures) for invited discussions, and *three* pages for contributed discussions. By August 23d, 2010, just after the summer vacation, authors of invited papers should have sent the local organizer a final manuscript, together with a rejoinder to the discussion of their paper. The final version should not exceed *twenty four* typeset pages (including figures and references), and the rejoinder should not exceed *six*. The \LaTeX source of any accepted contribution to the discussion, separate **eps** postscript files of all figures, and the **pdf** files of the output (to be used for control), should be sent by e-mail to **valenciameeting@uv.es**. All contributions are expected to use the purpose-built \LaTeX Valencia macros, which may be downloaded from the conference website **www.uv.es/valenciameeting**. Typeset proofs will be sent to authors as they become available. We expect the proceedings to be finalized and sent to the printers by November 2010.

Bayesian Analysis (**ba.stat.cmu.edu**) will publish selected contributed papers presented at the World Meeting (as talks or posters). Submissions are subject to the standard refereeing process of Bayesian Analysis. Please make sure to indicate in your cover letter that the paper was presented at ISBA 2010. All such papers received by August 16 will be eligible for the Lindley prize (**www.bayesian.org/awards/LindleyPrize.html**).

Acknowledgements

We are grateful to the Universitat de València for its continuous support, and to the US agencies National Science Foundation (NSF), National Institutes of Health (NIH) and Office of Naval Research Global (ONRG), for their partial funding.

Welcome to Spain!

José M. Bernardo

- **Marzouk, Youssef** (Massachusetts Institute of Technology, USA)
Paul Boggs (Sandia National Laboratories, USA)
Nonparametric Bayesian density estimation using polynomial chaos expansions
- **Mason, Alexina** (Imperial College London, UK)
Best, Nicky (Imperial College London, UK)
Richardson Sylvia (Imperial College London, UK)
Plewis, Ian (University of Manchester, UK)
Strategy for modelling non-random missing data mechanisms in longitudinal studies: Application to in-come data from the millennium cohort study
- **Mayoral, Asunción** (Universidad Miguel Hernández de Elche, Spain)
Quesada, M. (Universidad Miguel Hernández de Elche, Spain)
Morales, J. (Universidad Miguel Hernández de Elche, Spain)
Barber, Xavier (Universidad Miguel Hernández de Elche, Spain)
OBANSoft, a Bayesian software for objective Bayesian analysis: Its first days.
- **Mayrink, Vinicius** (Duke University, USA)
Lucas, Joseph (Duke University, USA)
A Bayesian factor model for gene expression detection on oligonucleotide microarrays.
- **McCandless, Lawrence** (Simon Fraser University, Canada)
Sequential versus full-Bayesian regression adjustment for the propensity score.
- **Mattei, Alessandra** (Università degli Studi di Firenze, Italy)
Mealli, Fabrizia (Università degli Studi di Firenze, Italy)
Bayesian inference in augmented designs for assessing principal strata causal effects
- **Mendoza, Manuel** (ITAM, Mexico)
Gutiérrez-Peña, Eduardo (IIMAS-UNAM, Mexico)
Some thoughts on the Bayesian robustness of location-scale models
- **Merl, Daniel** (Lawrence Livermore National Laboratory, USA)
Sequential clustering and anomaly detection for streaming data
- **Meyer, Renate** (University of Auckland, New Zealand)
Cai, Bo (University of South Carolina, USA)
Adaptive Metropolis-Hastings-within-Gibbs algorithms using copulae
- **Migon, Helio** (Universidade Federal do Rio de Janeiro, Brazil)
Salazar, Esther (Universidade Federal do Rio de Janeiro, Brazil)
Alves, Larissa (Universidade Federal do Rio de Janeiro, Brazil)
A Bayesian spatial model for panel time series data



OBANSoft, a Bayesian software for Objective Bayesian Analysis: its first days

Mayoral, A.M., Quesada, M., Morales, J. and Barber, X.

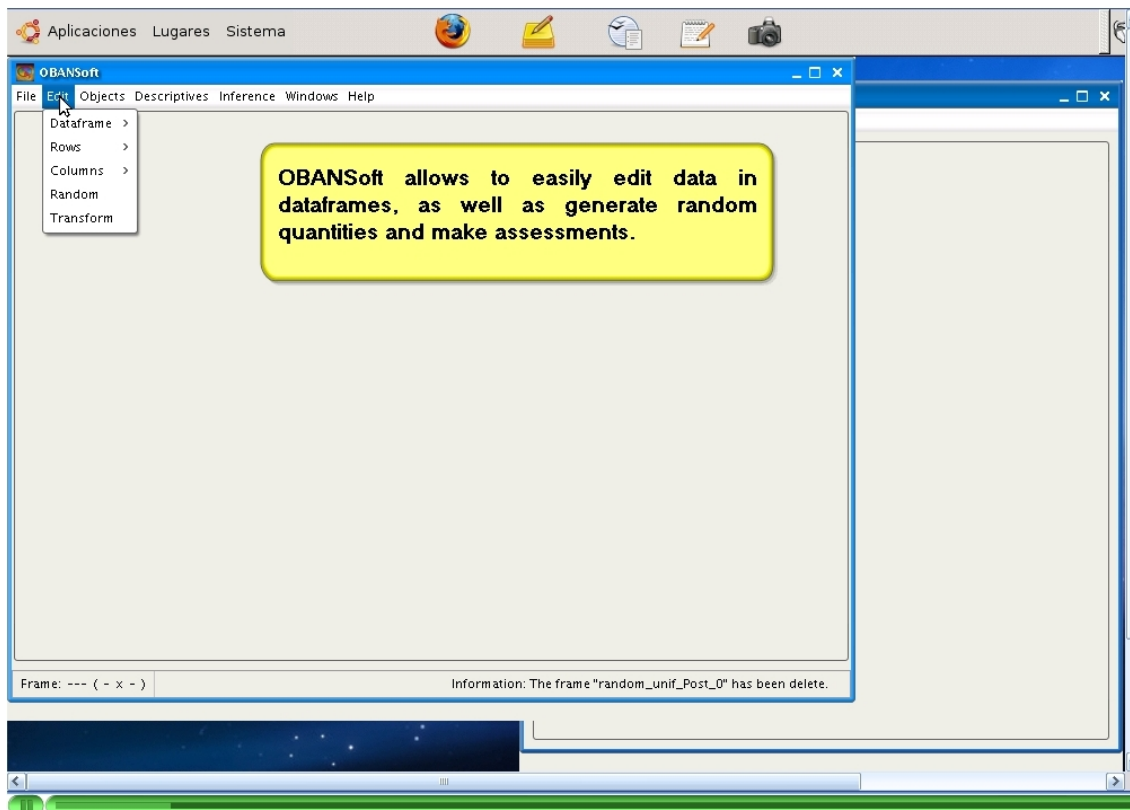
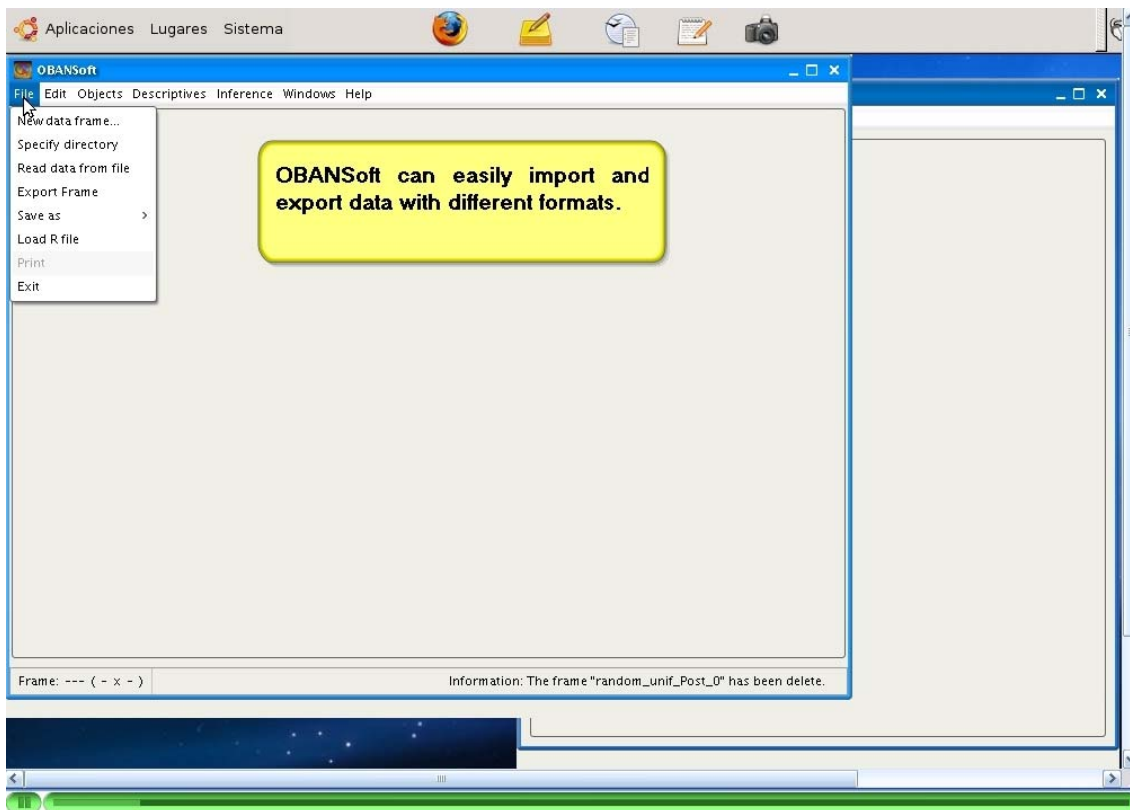
Universidad Miguel Hernández de Elche
SPAIN

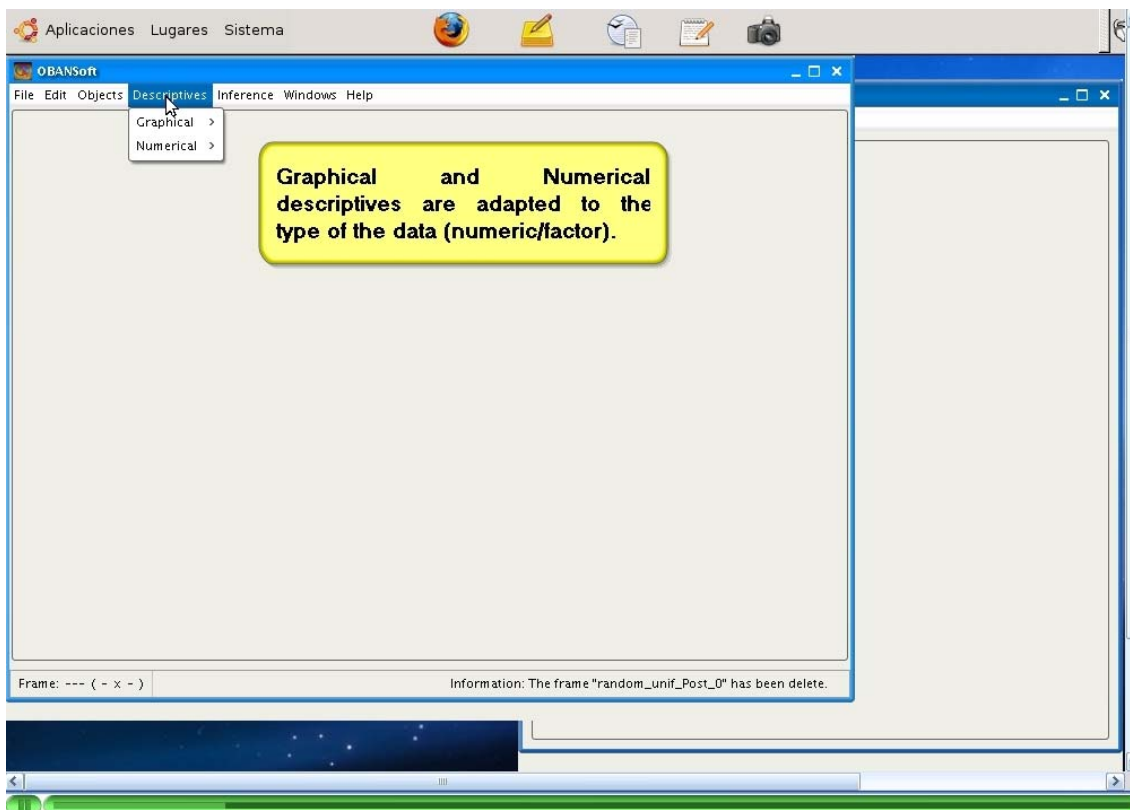
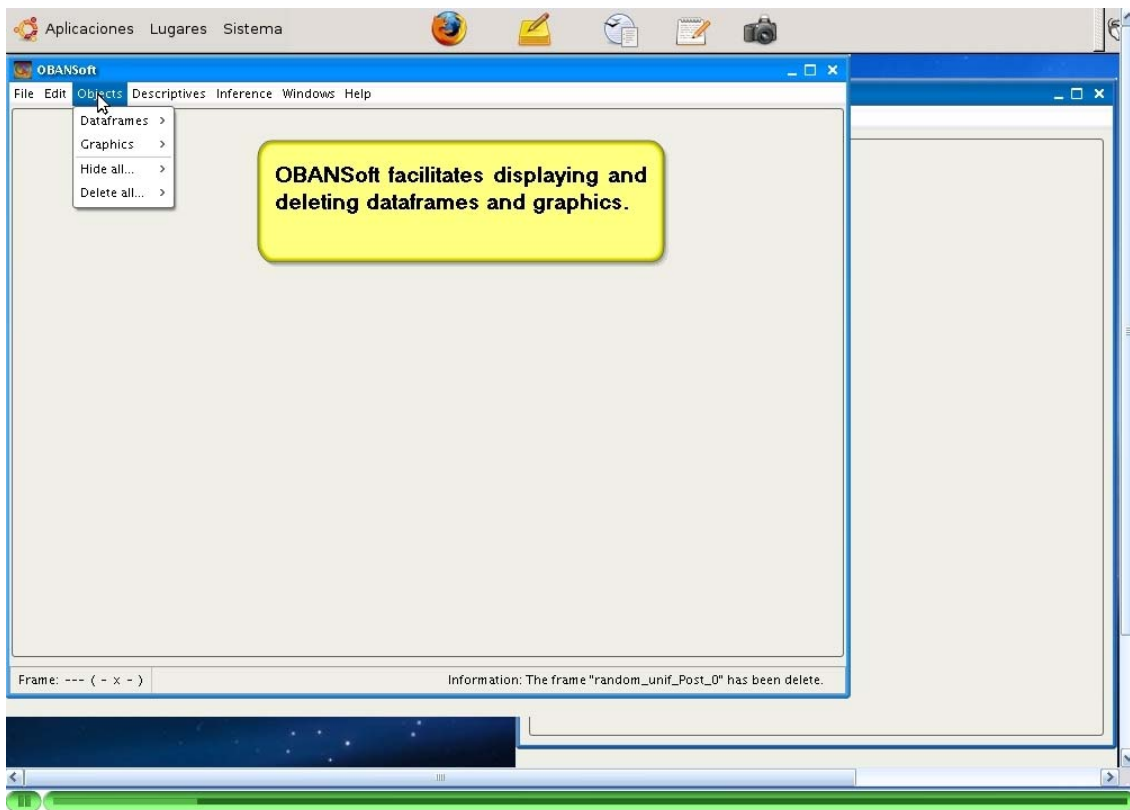


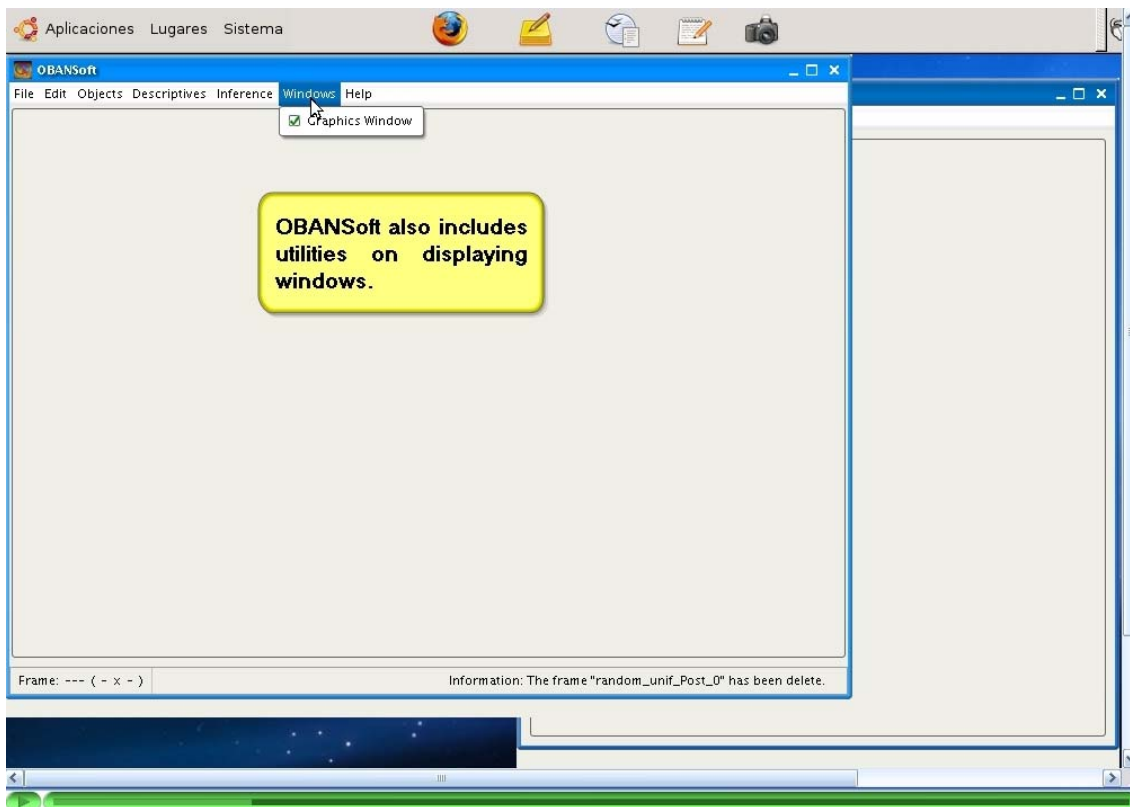
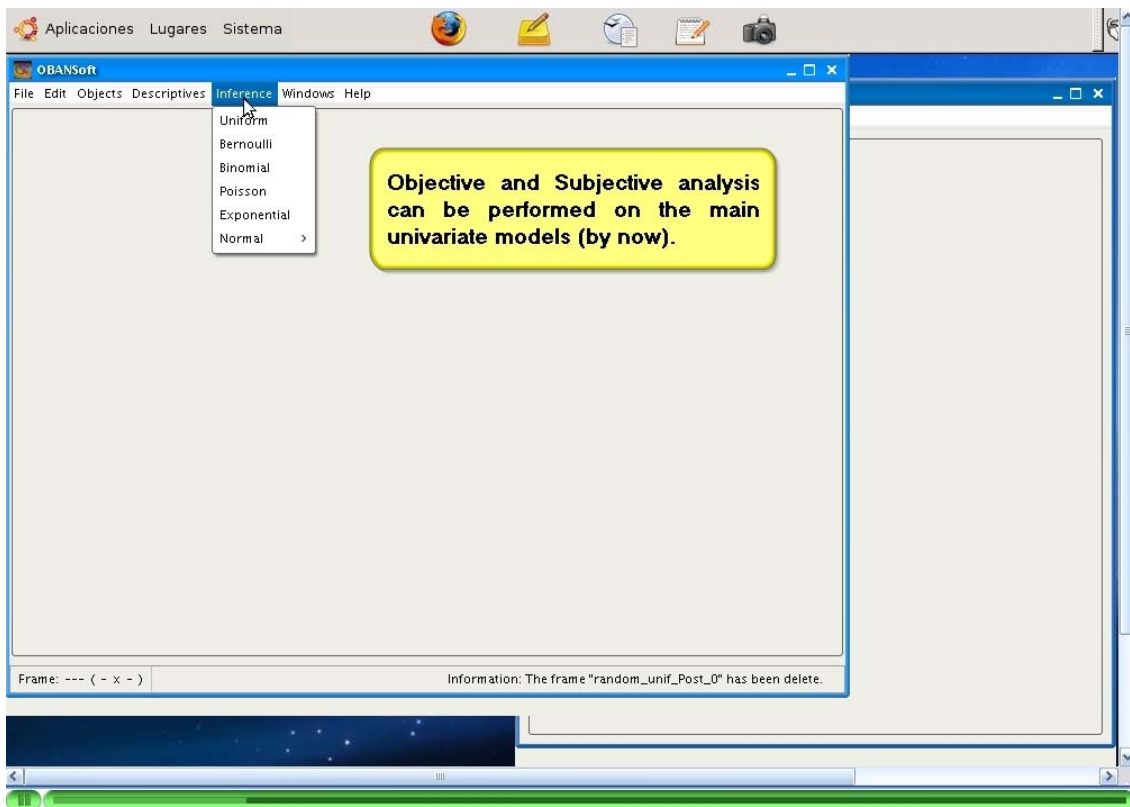
OBANSoft

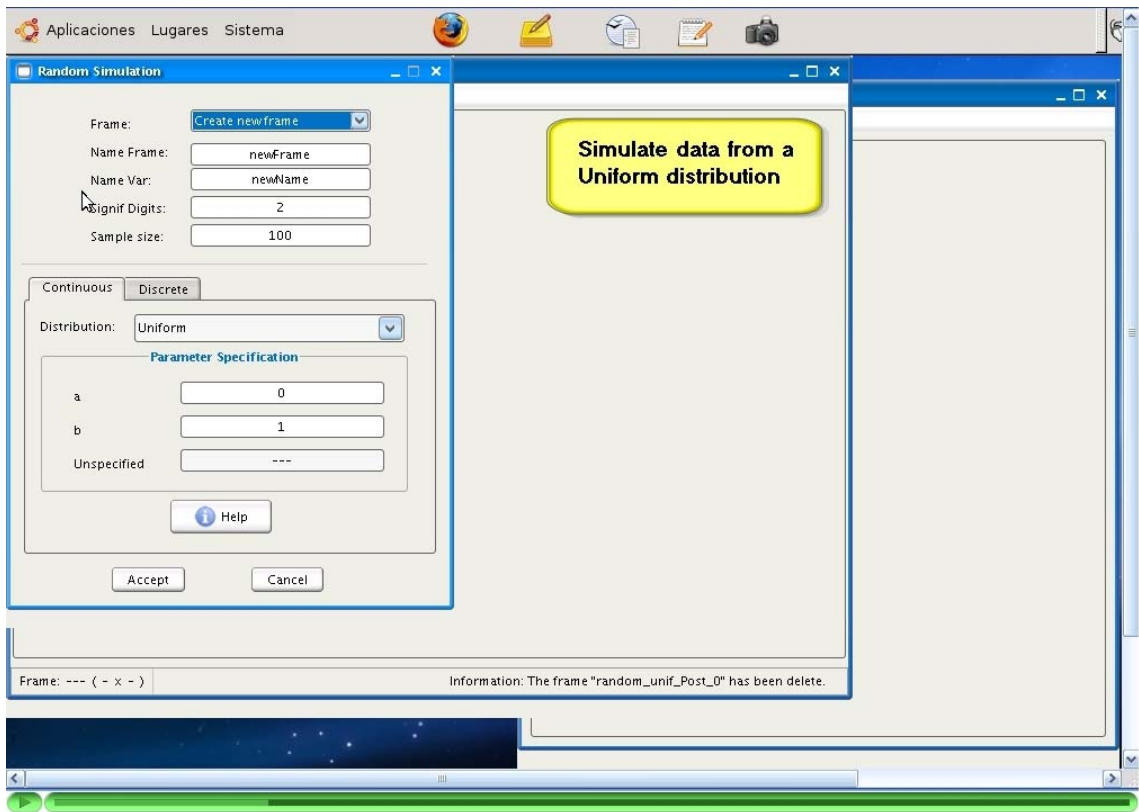
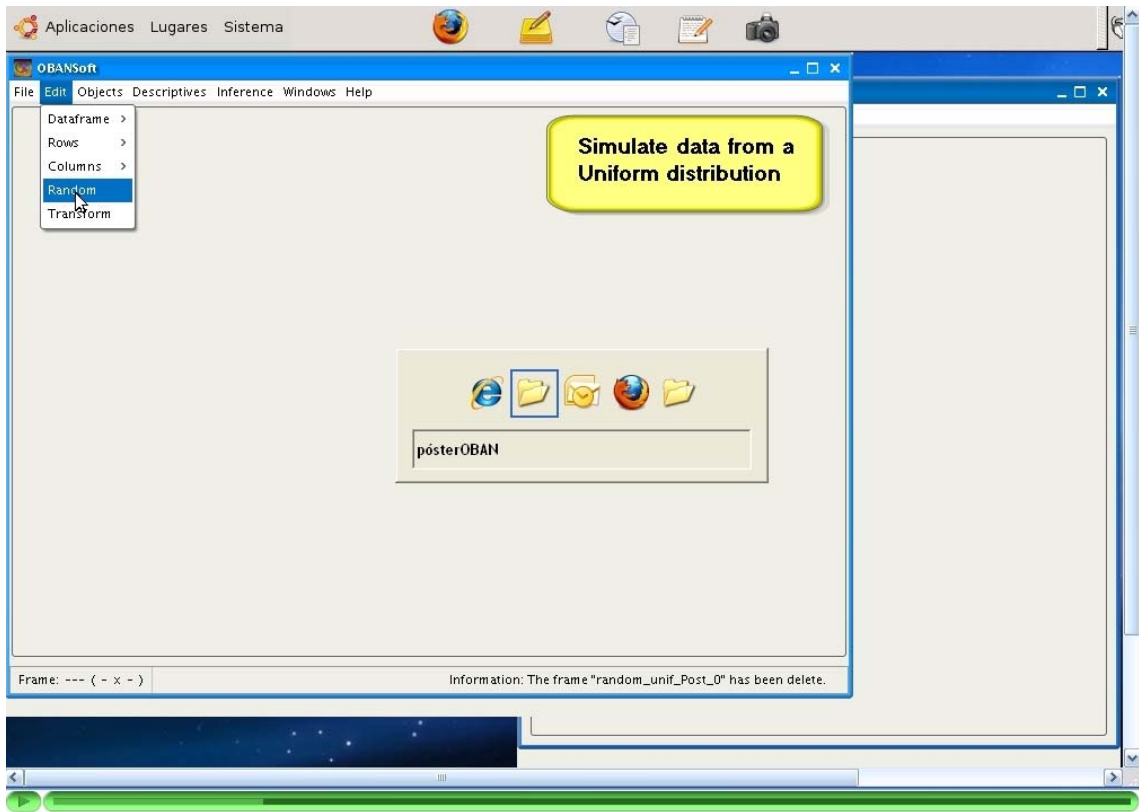
- Do provide students with a flexible tool to understand and experiment with basic concepts on Bayesian Statistics.
- Both perspectives, subjective and objective, are possible.
- Has been programmed under Java, with R as the calculus engine.
- A really kindly interface facilitates its use.

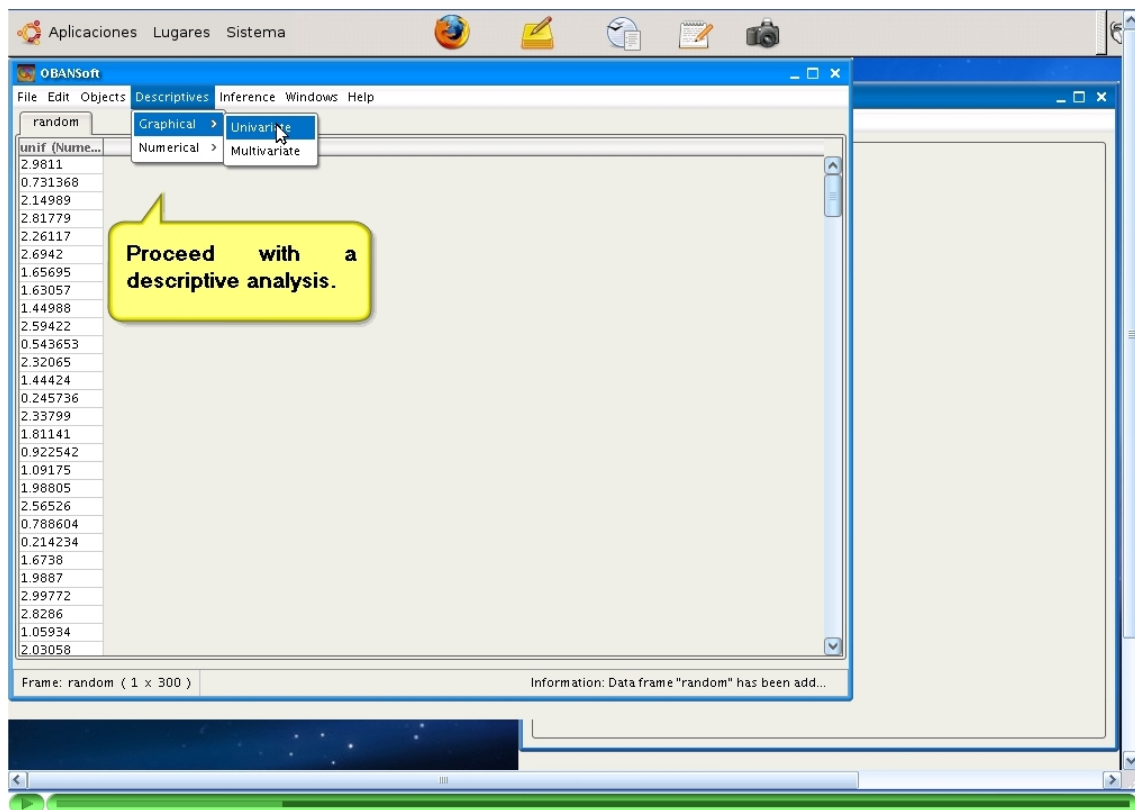
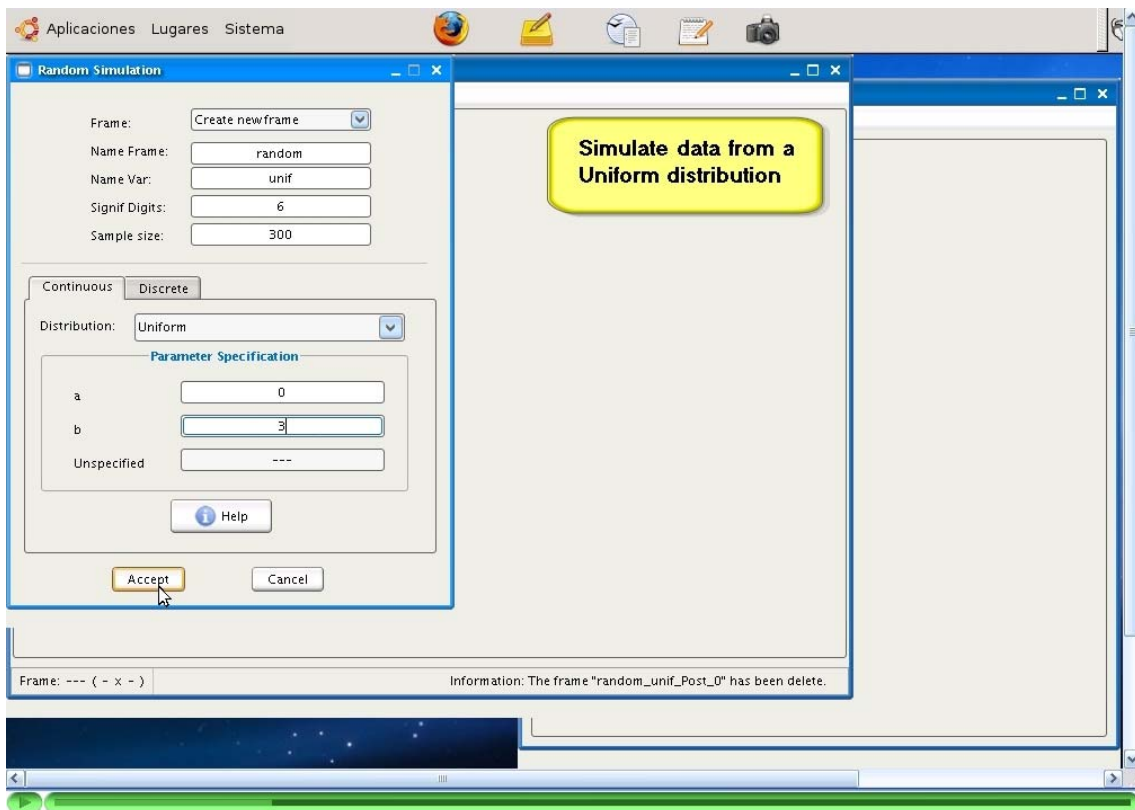


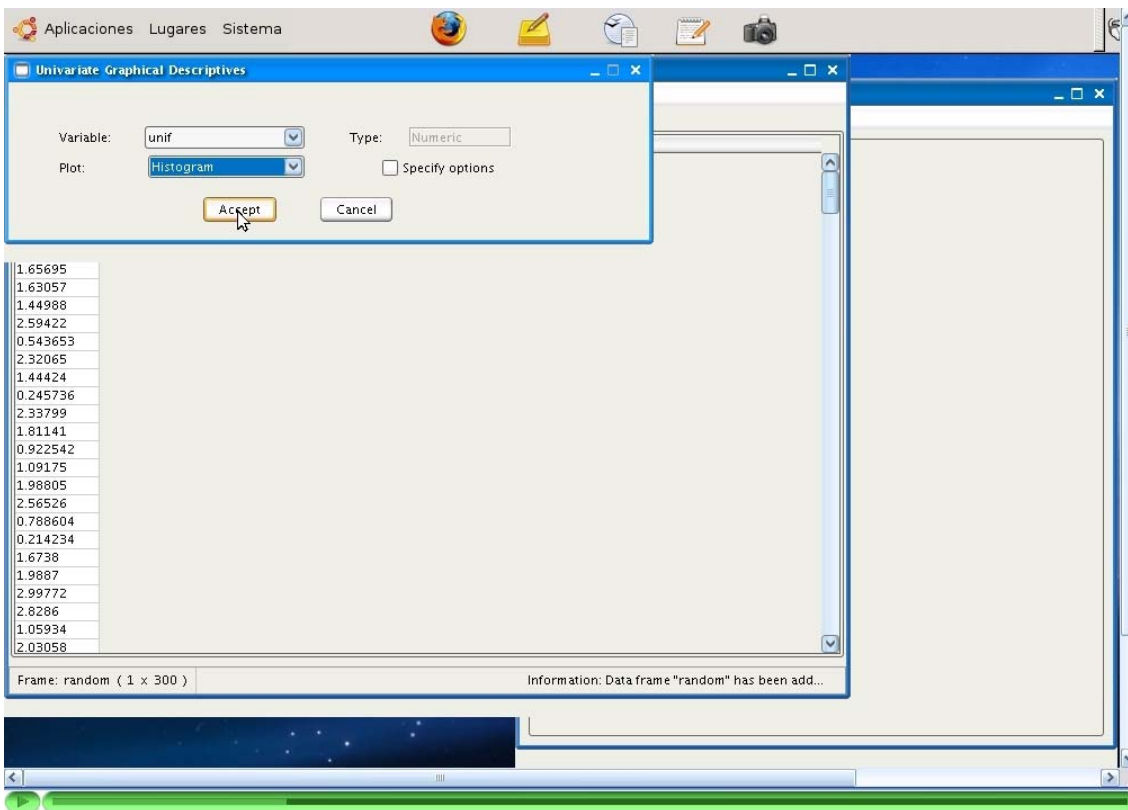
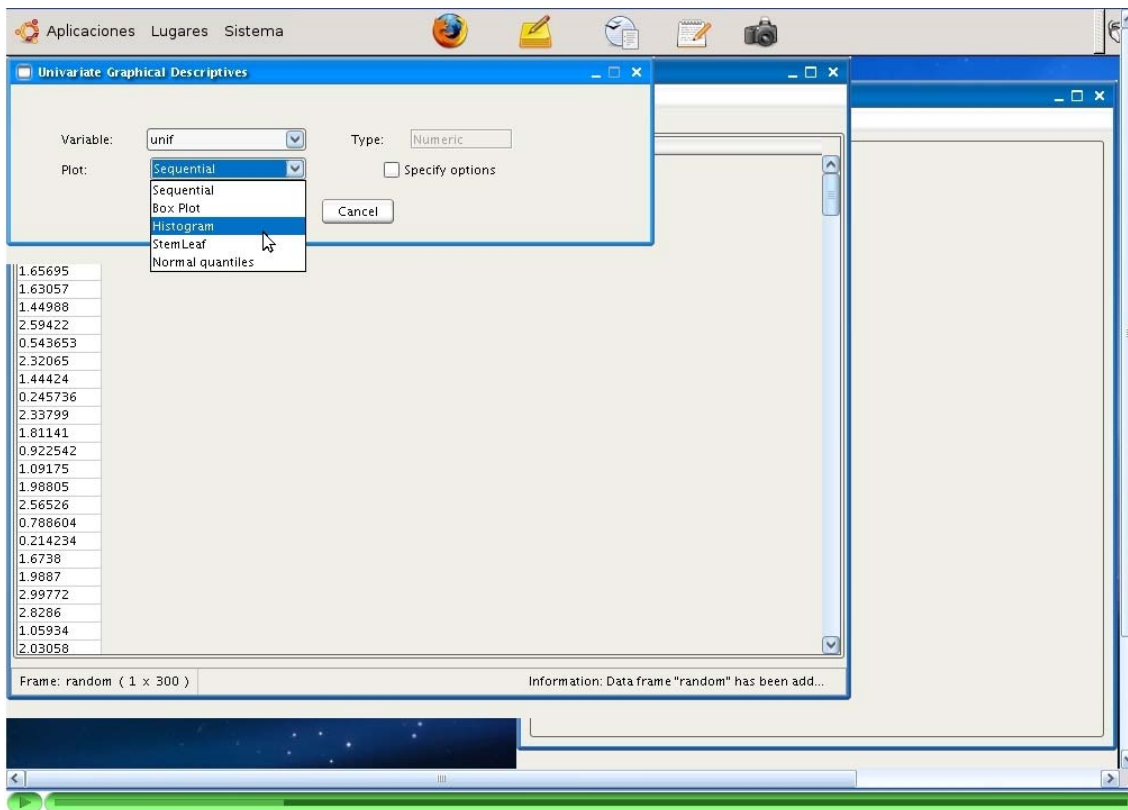


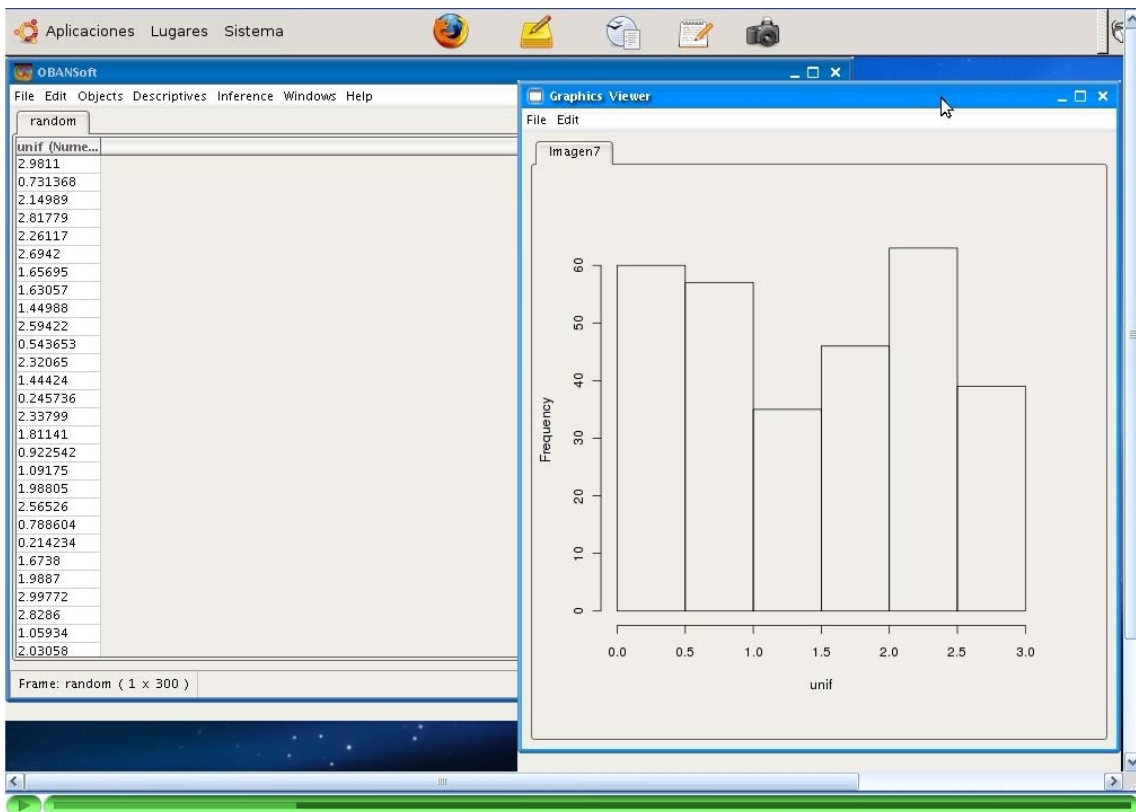
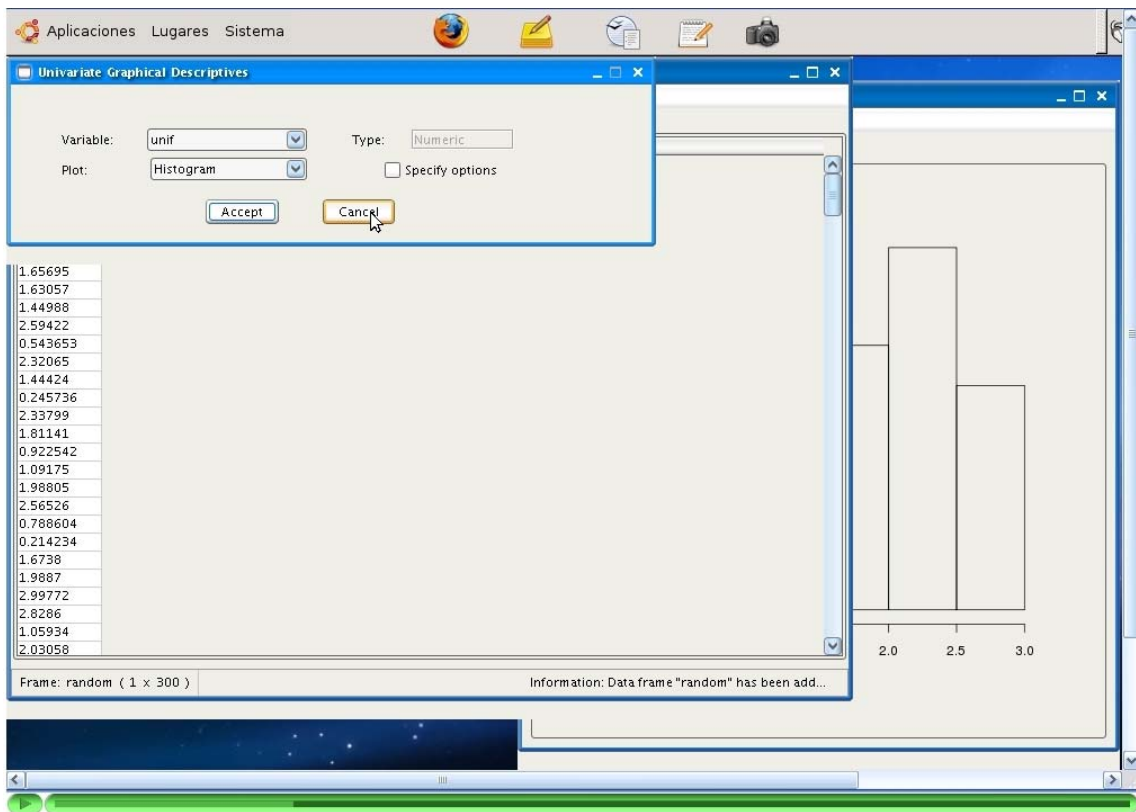












OBANSoft

File Edit Objects Descriptives Inference Windows Help

random

unif (Name...
 2.9811
 0.731368
 2.14989
 2.81779
 2.26117
 2.6942
 1.65695
 1.63057
 1.44988
 2.59422
 0.543653
 2.32065
 1.44424
 0.245736
 2.33799
 1.81141
 0.922542
 1.09175
 1.98805
 2.56526
 0.788604
 0.214234
 1.6738
 1.9887
 2.99772
 2.8286
 1.05934
 2.03058

Information: Data frame "random" has been add...

Frame: random (1 x 300)

Next, proceed with the analysis.

Uniform Model

Bayesian Analysis Help

Subjective analysis

Data

Frame: random

Variable: unif

Valid Cases: 300 Missing Values: 0

Distribution: Pareto

Parameters:

alpha: 2

beta: 3

Unspecified: ---

Predictive Analysis

Type: Prior Predictive

Simulation Options:

Nsim: 1000

Burn: 0

Nchains: 1

Go

Posterior Inference

Parameter: theta

Transformation: < Double click >

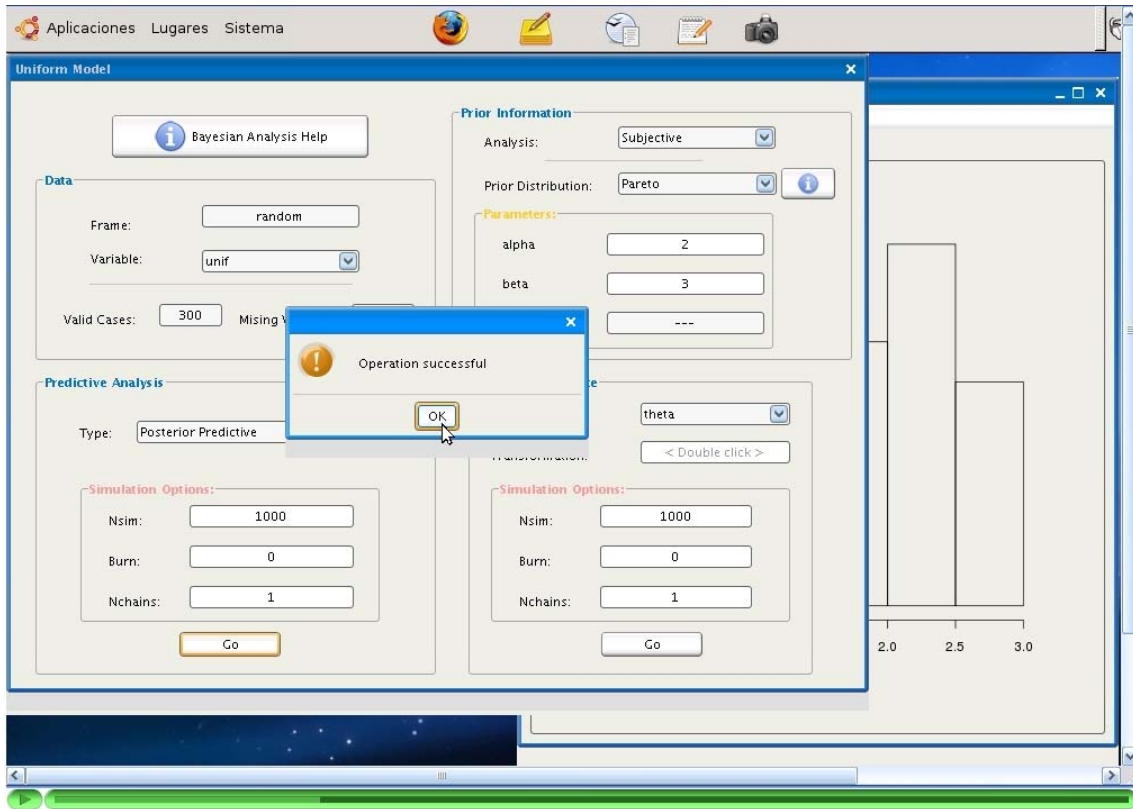
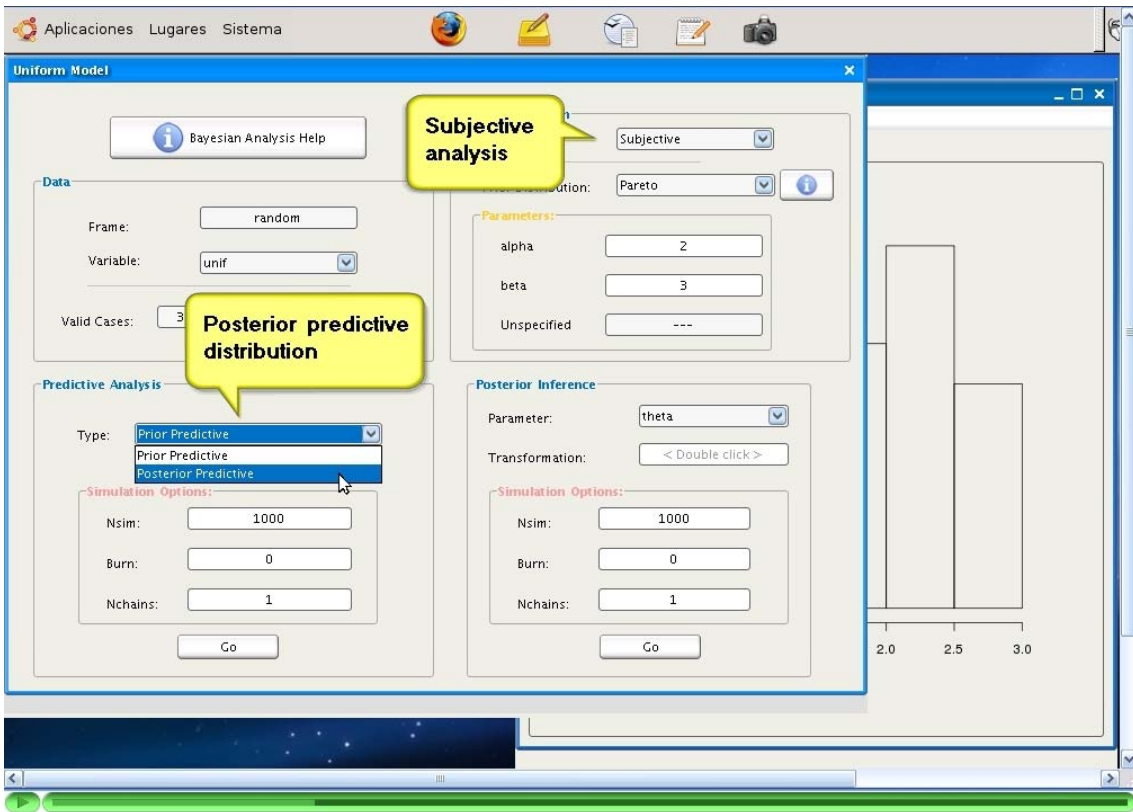
Simulation Options:

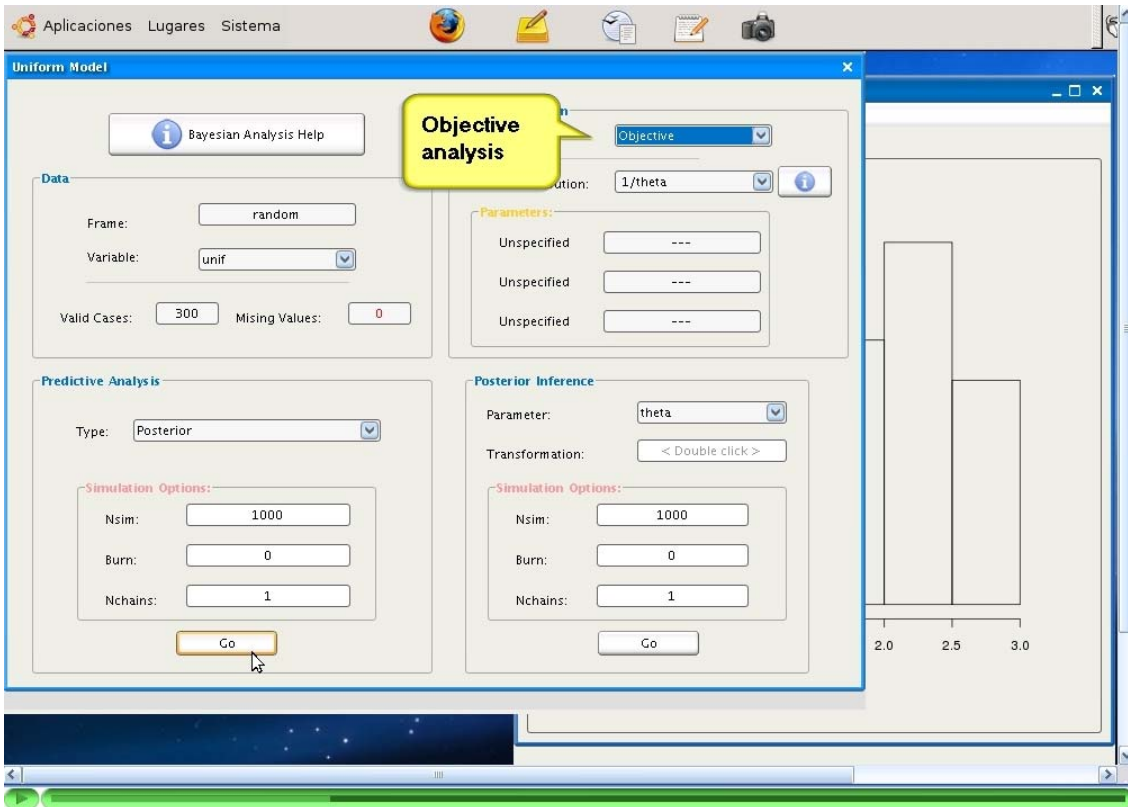
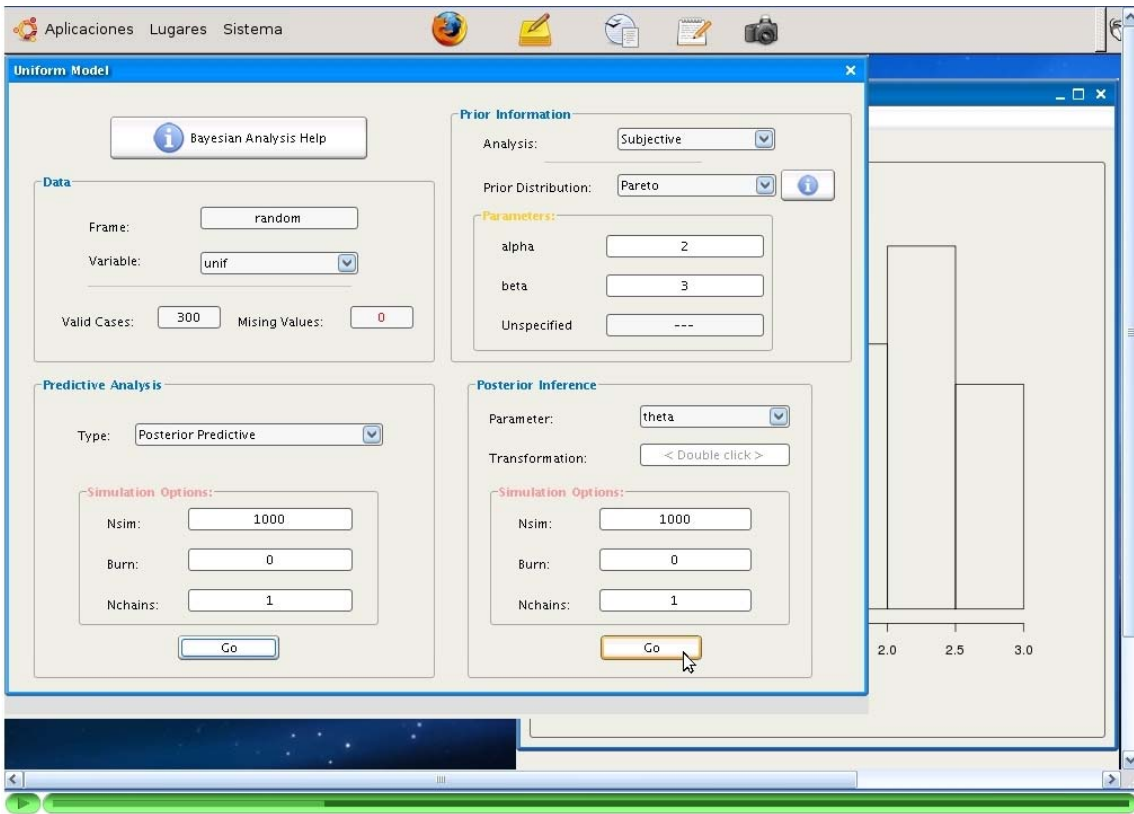
Nsim: 1000

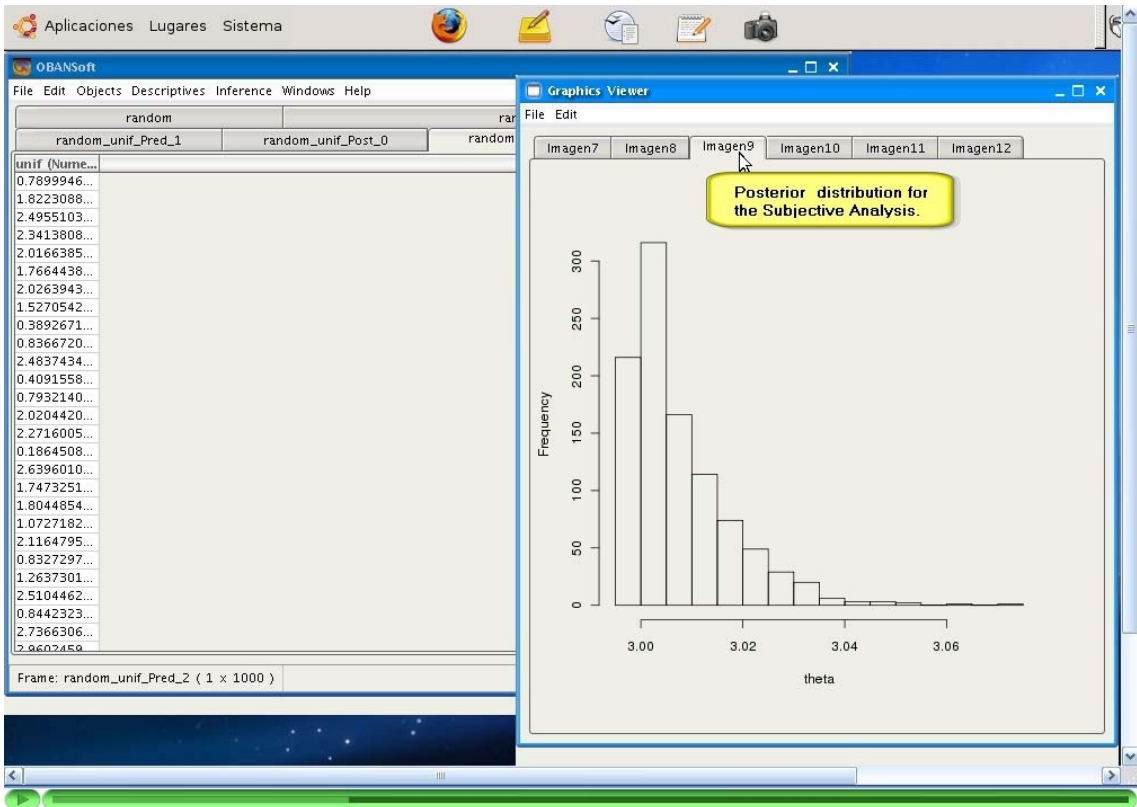
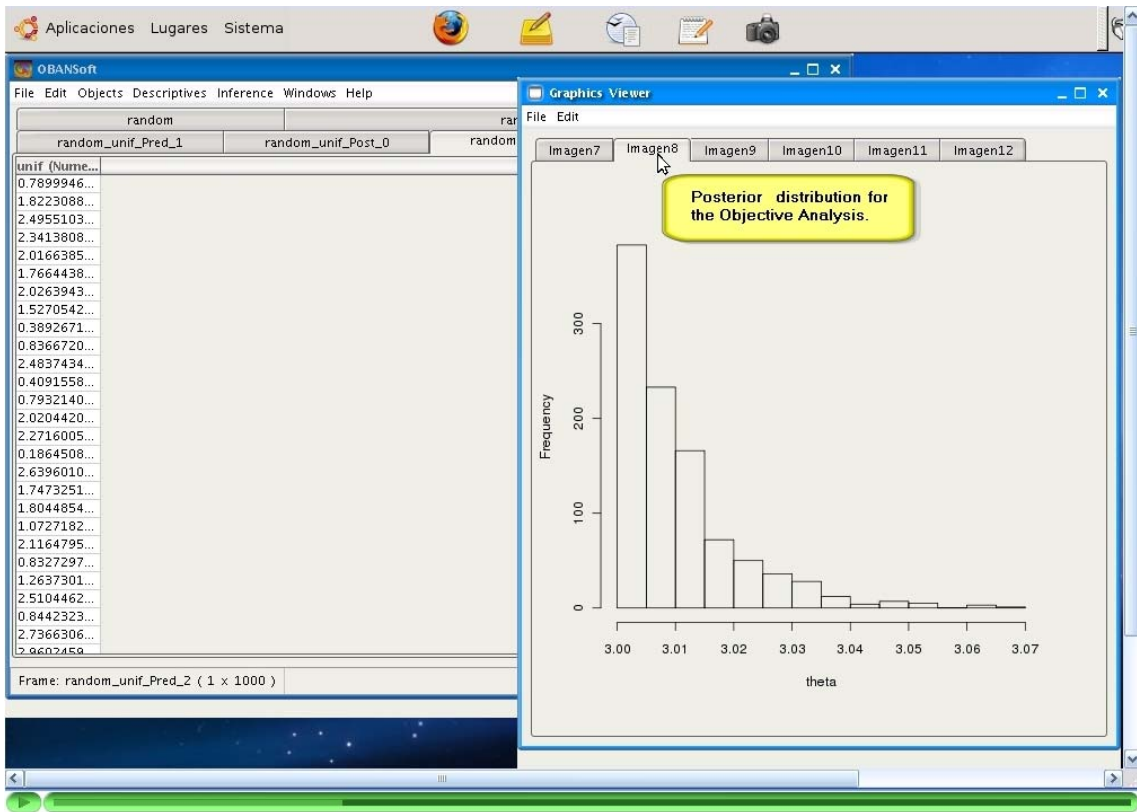
Burn: 0

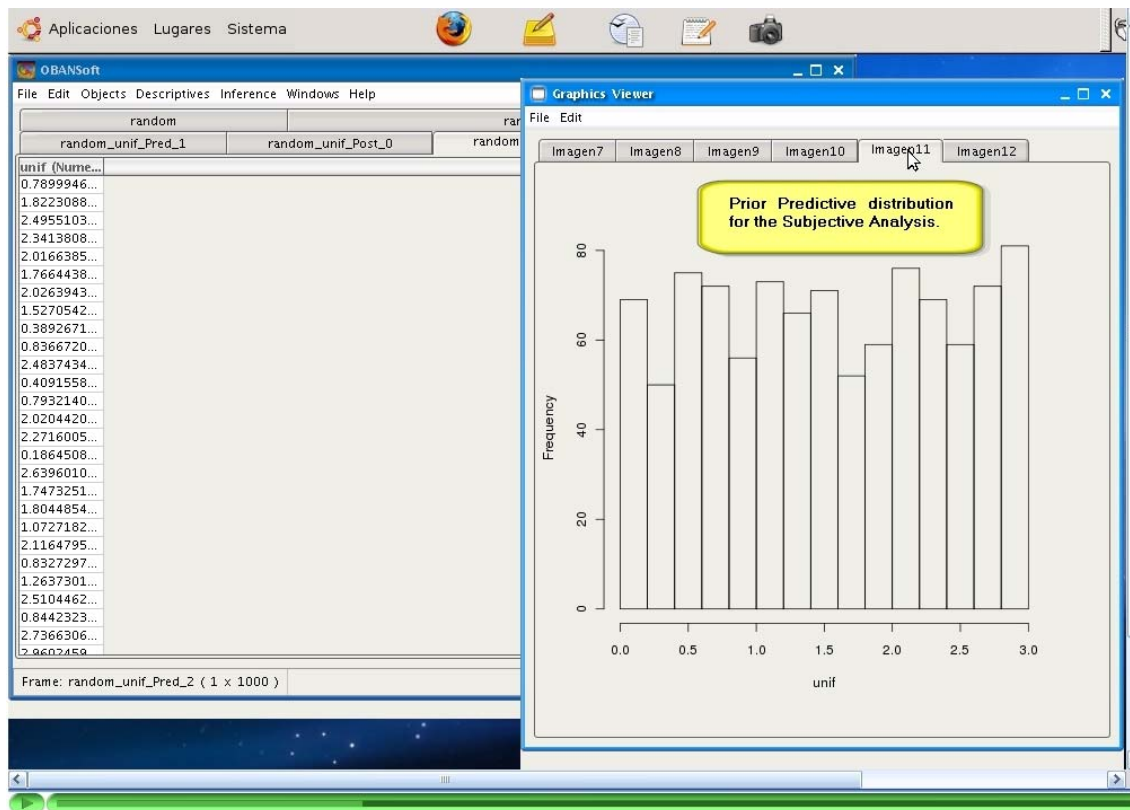
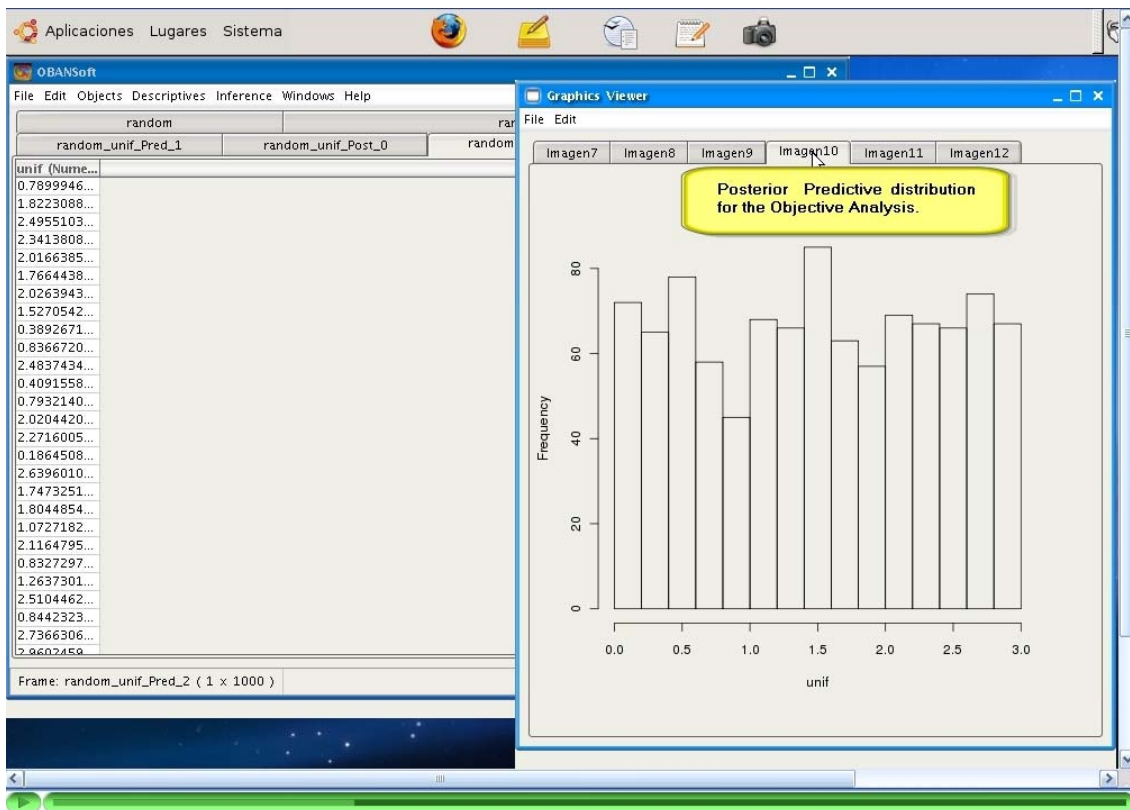
Nchains: 1

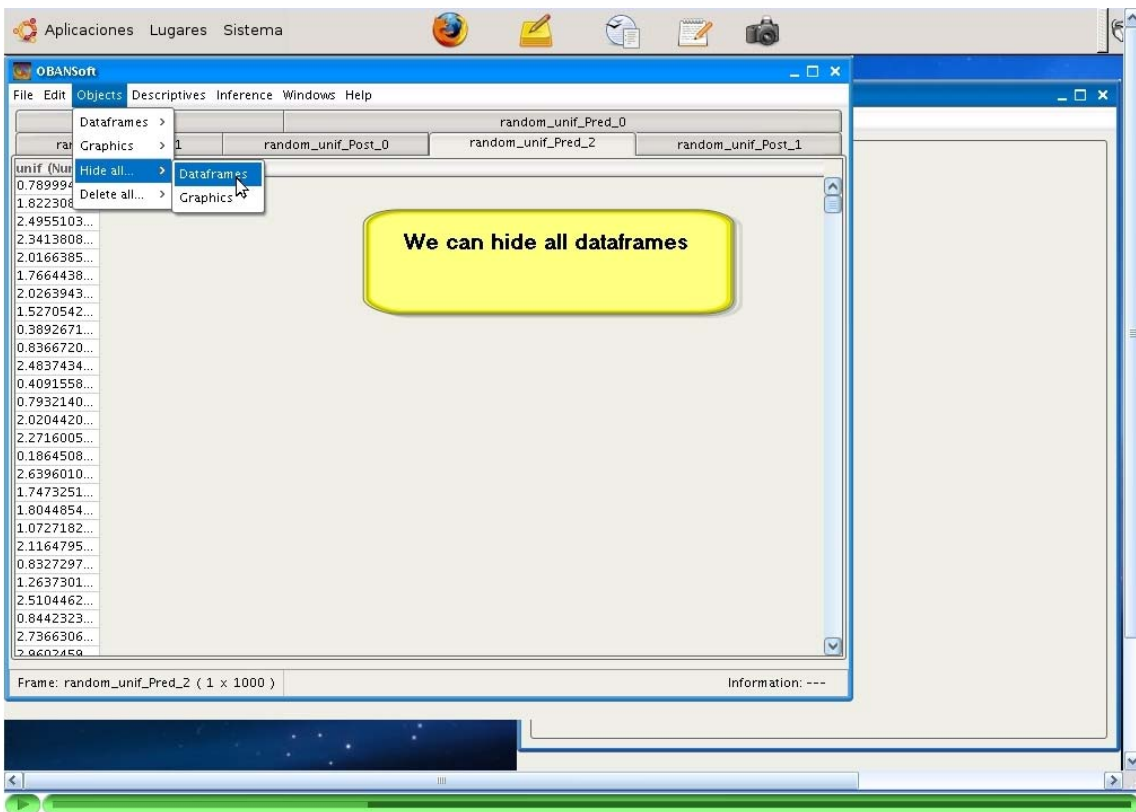
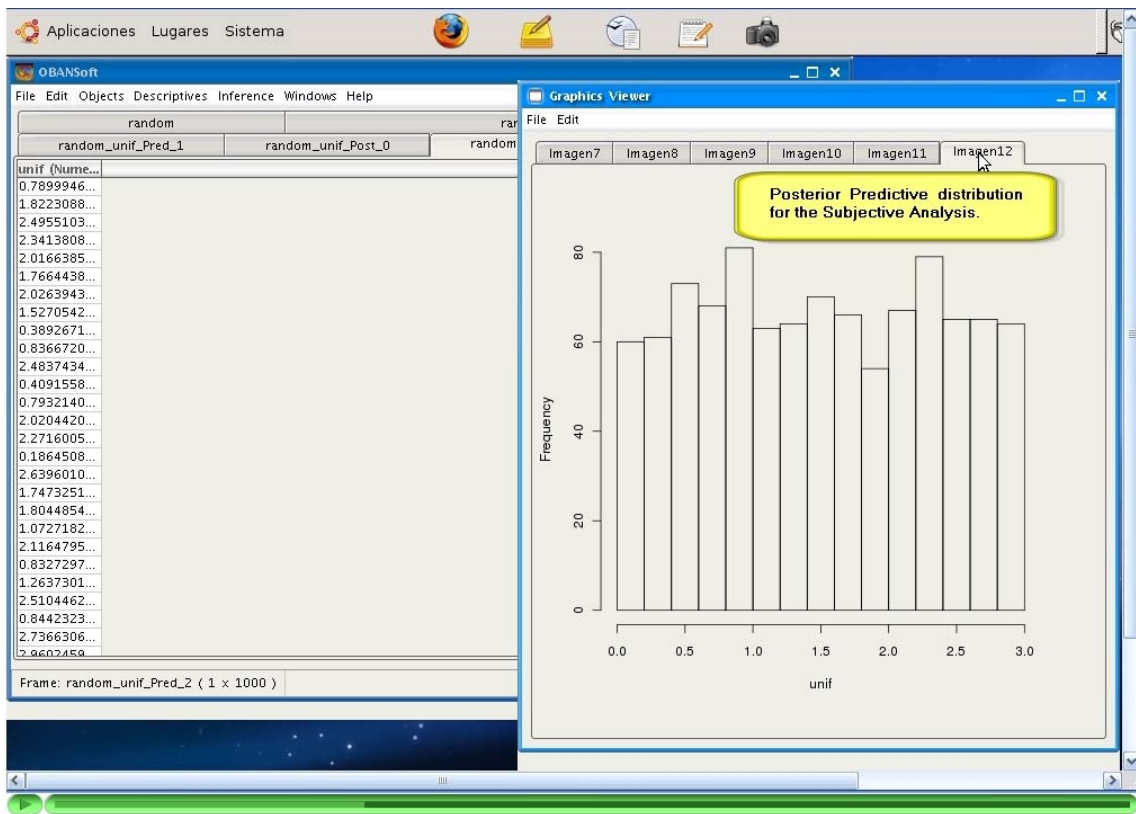
Go

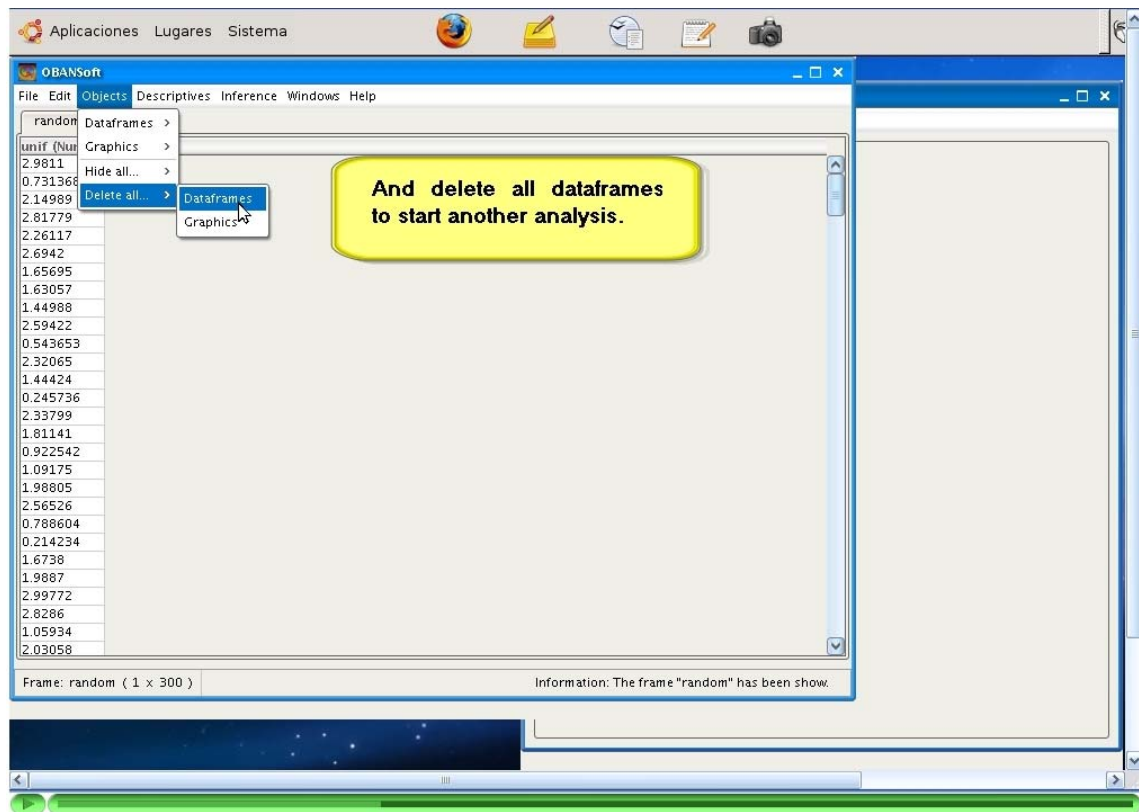
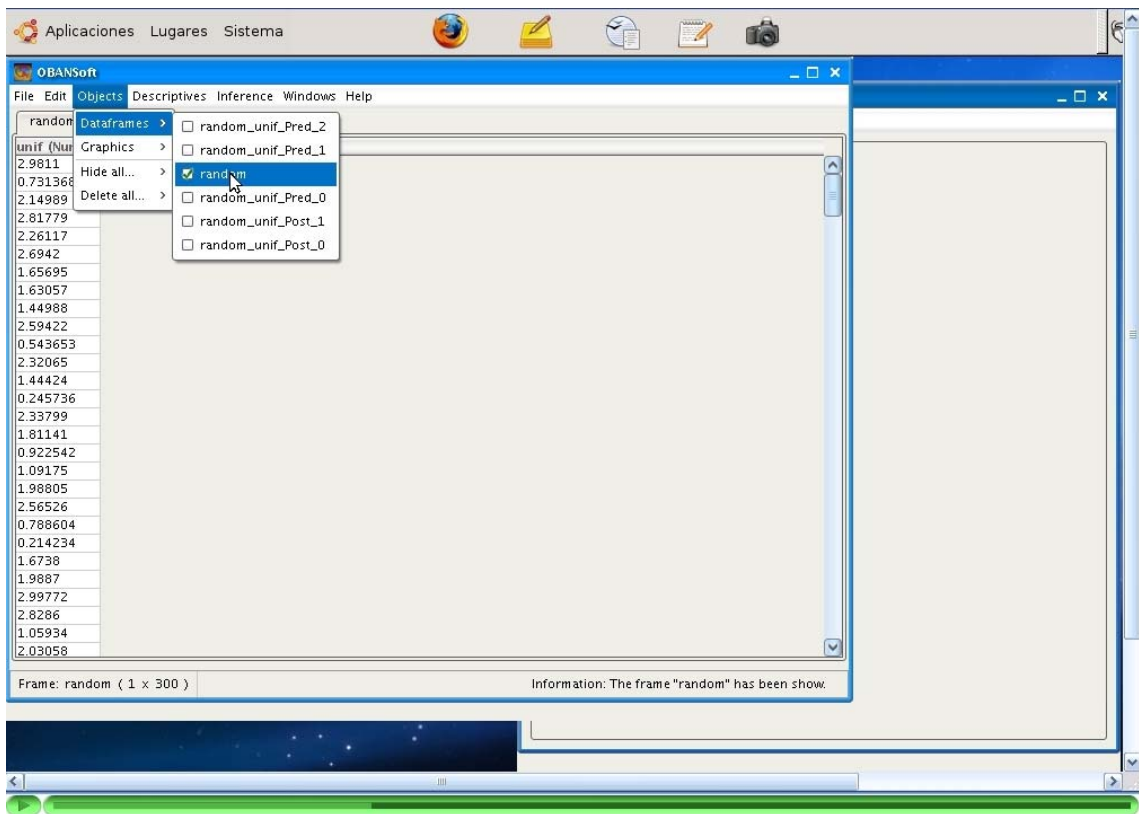


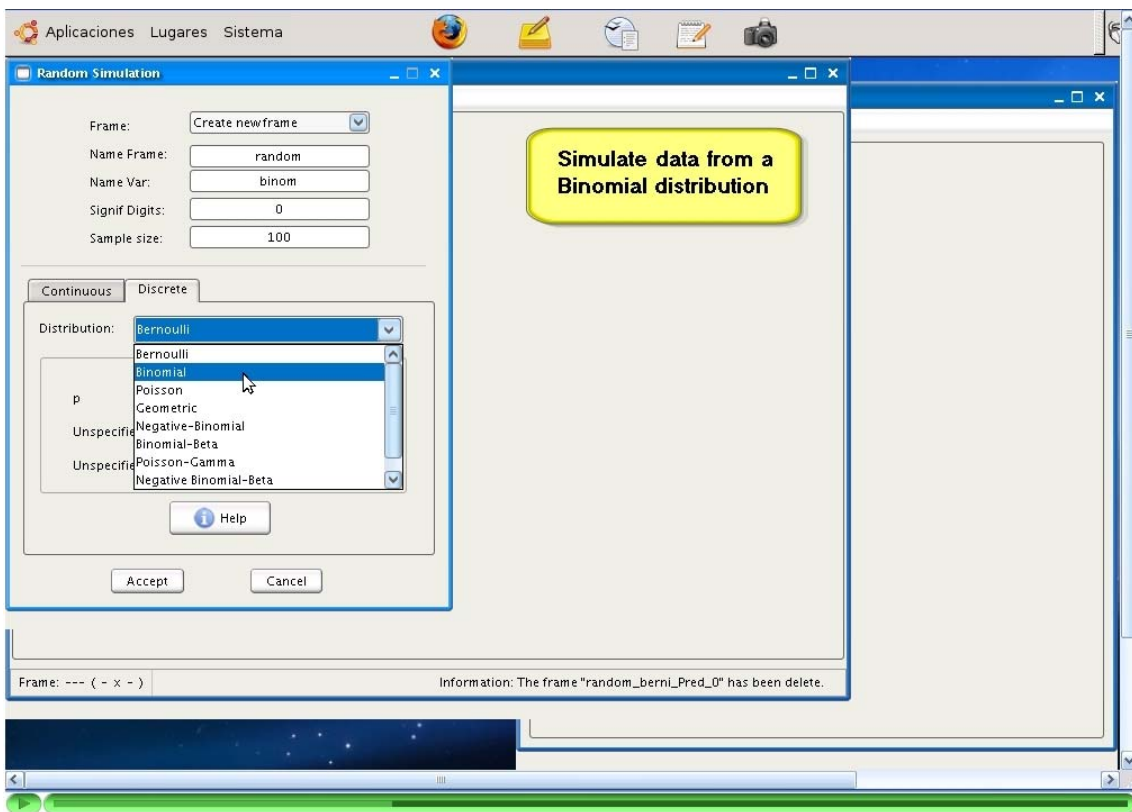
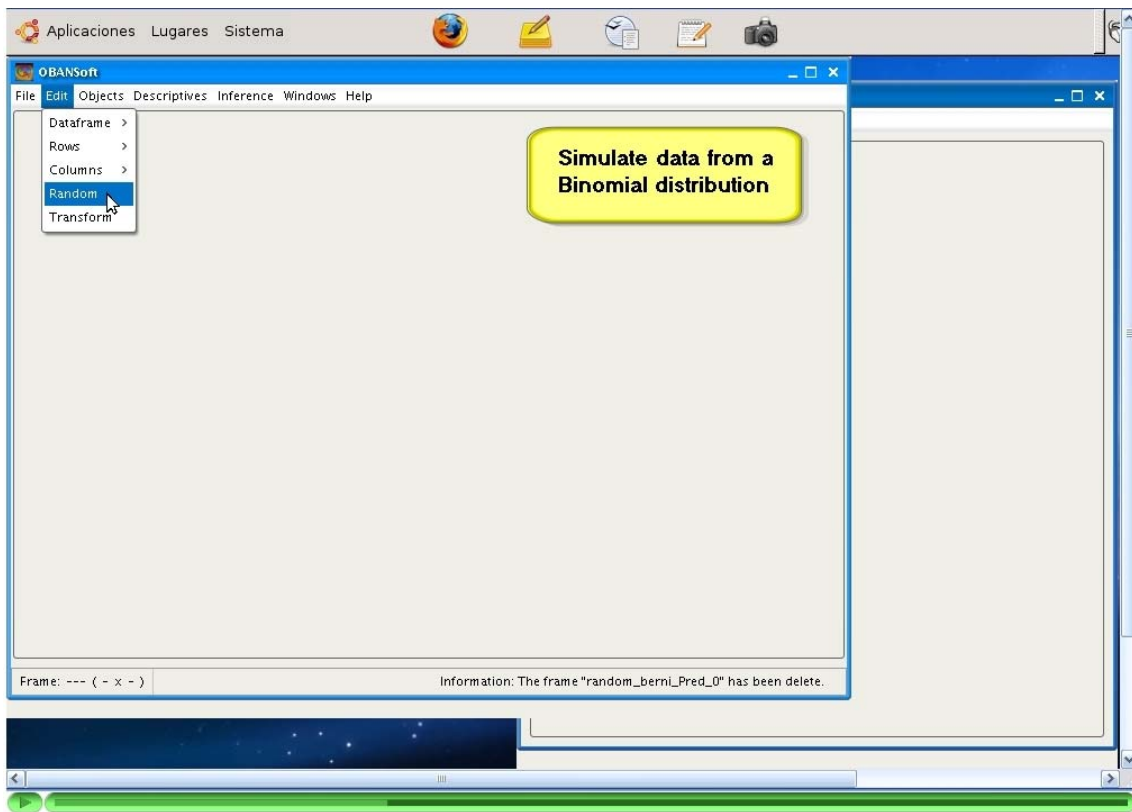


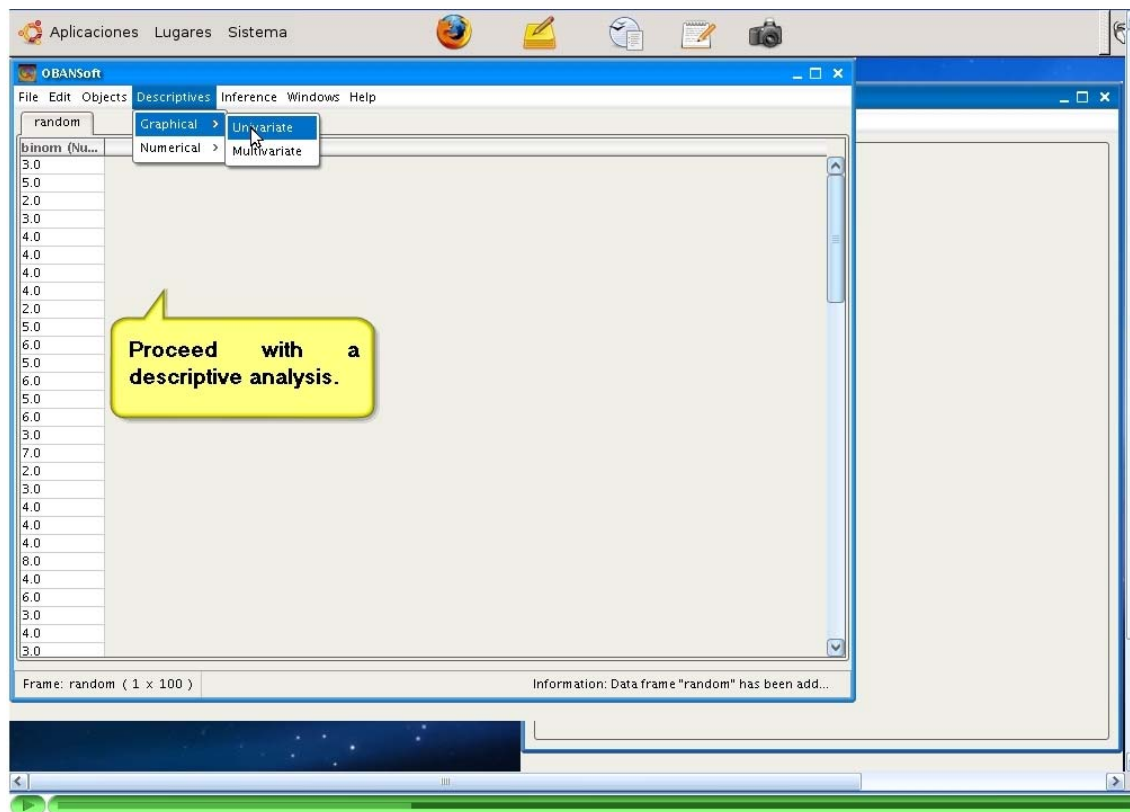
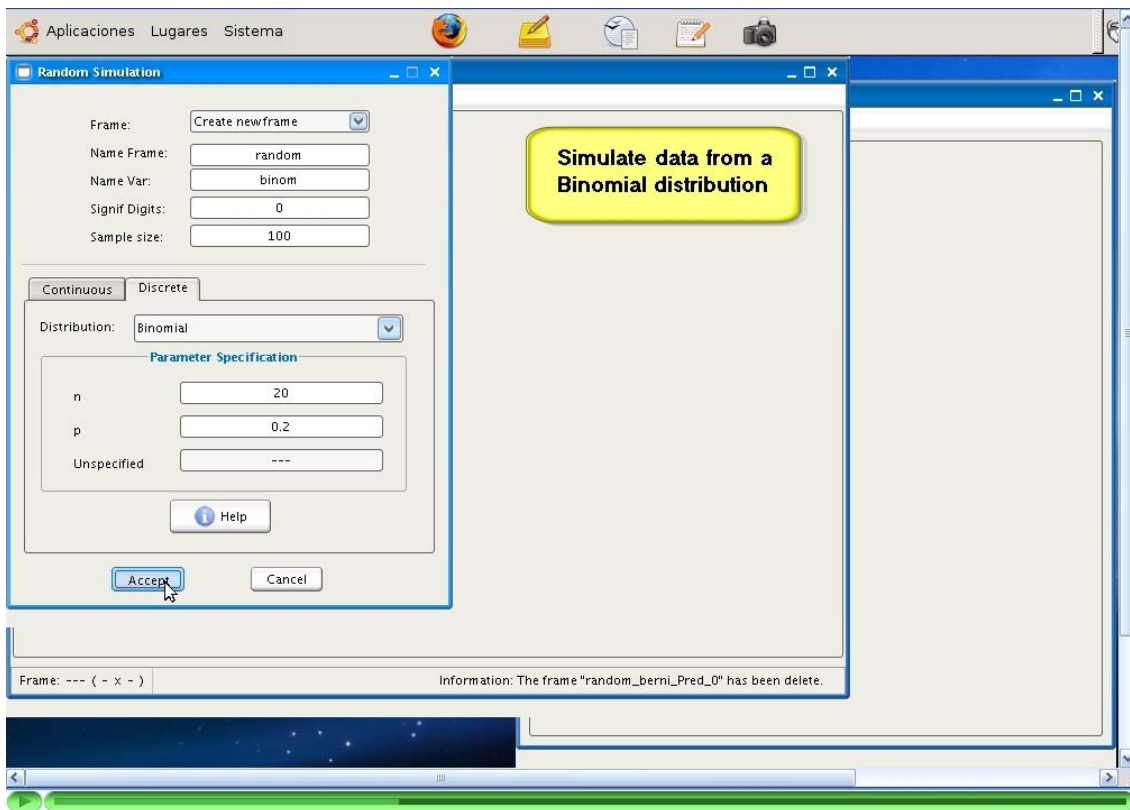












Univariate Graphical Descriptives

Variable: binom Type: Numeric

Plot: Sequential Specify options

Cancel

Proceed with a descriptive analysis.

Frame: random (1 x 100) Information: Data frame "random" has been add...

OBANSoft

File Edit Objects Descriptives Inference Windows Help

random

binom (Nu...

3.0
5.0
2.0
3.0
4.0
4.0
4.0
2.0
5.0
6.0
6.0
5.0
6.0
3.0
7.0
2.0
3.0
4.0
4.0
8.0
4.0
6.0
3.0
4.0
3.0

Frame: random (1 x 100)

Graphics Viewer

File Edit

Imagen14

Frequency

binom

binom	Frequency
1	6
2	9
3	13
4	23
5	21
6	9
7	5
8	3
9	1

OBANSoft

File Edit Objects Descriptives Inference Windows Help

random

- Rename
- Convert
 - To Character
 - To Factor
 - To Numeric
 - To Ordered
- Sort
 - To Factor
- Factor labels
 - To Numeric
 - To Ordered
- Remove column

Frame: random (1 x 100)

Information: Data frame "random" has been add...

But we can change the features of the variable, to access categorical descriptives

OBANSoft

File Edit Objects Descriptives Inference Windows Help

random

- Graphical
 - Univariate
 - Multivariate
- Numerical

binom (Fac...

Frame: random (1 x 100)

Information: Data frame "random" has been add...

But we can change the features of the variable, to access categorical descriptives

Univariate Graphical Descriptives

Variable: binom Type: Numeric

Plot: Barplot

Specify options

Frame: random (1 x 100) Information: Data frame "random" has been add...

4
4
2
5
6
5
6
5
6
3
7
2
3
4
4
4
8
4
6
3
4
3

6 8

Univariate Graphical Descriptives

Variable: binom Type: Numeric

Plot: Barplot

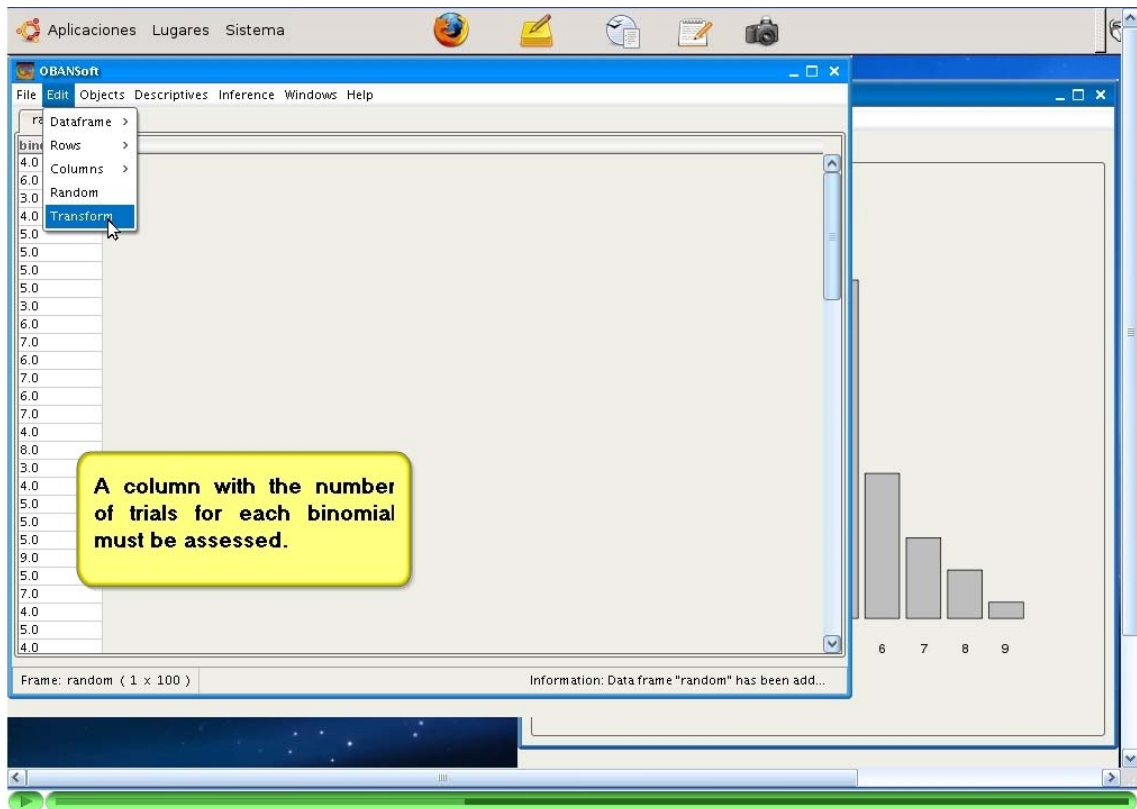
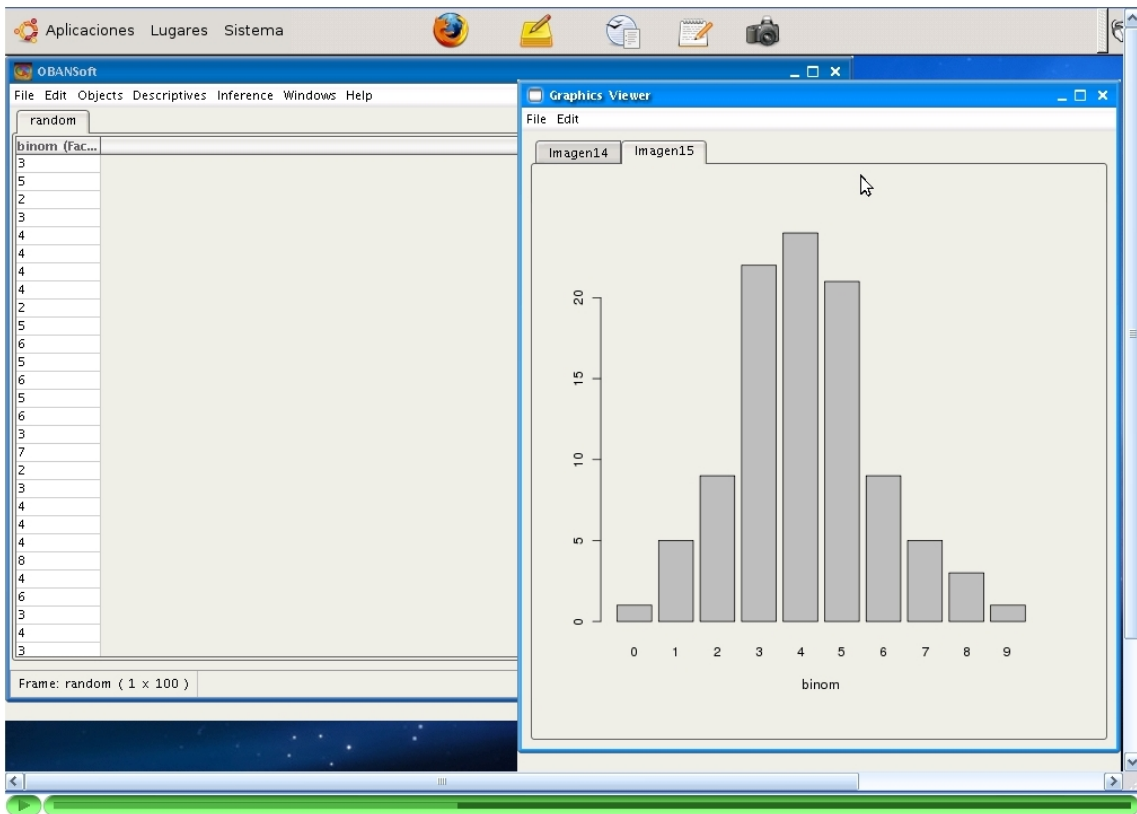
Specify options

Accept Cancel

Frame: random (1 x 100) Information: Data frame "random" has been add...

4
4
2
5
6
5
6
5
6
3
7
2
3
4
4
4
8
4
6
3
4
3

6 8



Transform Numerical Variables

Numeric Variables: binom

Frame: random | Newvariable: ntrials

Transform: 20

Elemental functions: sqrt(V1), exp(V1), log(V1)

Random noise: Unif(0,1), Exp(mean)

Buttons: Accept, Cancel

A column with the number of trials for each binomial must be assessed.

Frame: random (1 x 100) | Information: Data frame "random" has been add...

OBANSoft

File Edit Objects Descriptives Inference Windows Help

random

binom (Nu... | ntrials (Nu...)

4.0 | 20.0

6.0 | 20.0

3.0 | 20.0

4.0 | 20.0

5.0 | 20.0

5.0 | 20.0

5.0 | 20.0

3.0 | 20.0

6.0 | 20.0

7.0 | 20.0

6.0 | 20.0

7.0 | 20.0

6.0 | 20.0

7.0 | 20.0

4.0 | 20.0

8.0 | 20.0

3.0 | 20.0

4.0 | 20.0

5.0 | 20.0

5.0 | 20.0

9.0 | 20.0

5.0 | 20.0

7.0 | 20.0

4.0 | 20.0

5.0 | 20.0

4.0 | 20.0

Inference menu: Uniform, Bernoulli, **Binomial**, Poisson, Exponential, Normal >

Next, proceed with the analysis.

Frame: random (1 x 100) | Information: Data frame "random" has been add...

Se encuentra disponible el tutorial completo con los demás modelos de inferencia en la carpeta “Tutorial” del Cd adjunto.