Instituto Tecnológico de Costa Rica
IC-4700 Lenguajes de Programación
Profesor: Eddy Ramírez
I Semestre 2014

Documentación
Generando Funciones

Estudiantes:
Leonardo Madrigal Valverde
José Andrés García Sáenz

Tabla de Contenidos

1. Descripción del Problema

Se debe desarrollar un programa en scheme que genere números aleatorios, por medio de distribuciones de probabilidades y además otro programa en Java que posteriormente representará el gráfico de la distribución que se eligió para generar los números.
Los números generados en Scheme deben ser enviados a el programa en Java para la generación del gráfico. La comunicación entre los programas debe ser por medio de sockets y debe ser posible graficar distribuciones discretas y continuas.
Las distribuciones de probabilidades requeridas son: Tabla de Probabilidad, Poisson, Binomial, Geométrica, Hipergeométrica, Uniforme Discreta, Uniforme Continua, Exponencial, Normal y además distribuciones creadas por el usuario.

2. Método de Solución

En el módulo de Scheme se implementaron las funciones que se mencionaron en el punto anterior, además de otras funciones auxiliares que son necesarias para el funcionamiento correcto del programa, como las siguientes:

- Factorial: Calcula el factorial de un número.
- Combinaciones: Función que calcula el número de formas en que se pueden extraer k elementos de un conjunto de cardinalidad n.
- Método de Simpson: Calcula la integral definida de una función.
- Acumulada: Función que calcula la tabla acumulada de una distribución.
- Buscar en tabla: Función que se encarga de buscar, en una tabla acumulada de una distribución, una probabilidad.
- Generar tabla discreta: Genera la tabla de probabilidad de una función de distribución discreta en un rango definido.
- Generar tabla contínua: Genera la tabla de probabilidad de una función de distribución contínua en un rango definido con un intervalo.
- Funciones de envío de datos por socket: Se implementó una función, por cada tipo de distribuciones, encargada de enviar datos por socket.
- Controladores de distribuciones: Se crearon por cada tipo de distribución, una función encargada de controlar lo que se debe hacer para cada distribución. Esto es hacer el llamado de funciones auxiliares para cada distribución.
- Función “switch”: Se encarga de hacer el llamado al controlador respectivo de cada tipo de distribución.
- Una función de inicio: Se encarga de inicializar el socket, leer el archivo y hacer el llamado a la función “switch”.

El módulo de Java está diseñada en tres capas: una capa de conexión, una capa de utilidades para el sistema y la capa de interfaz gráfica. Las clases creadas por cada capa son las siguientes:

Capa de Conexiones:

- ClientConnectedEvent: Esta clase funciona como un contenedor de los datos del cliente conectado al servidor. Los datos disponibles son: El cliente que se conectó y la hora en que se conectó.
- DataRecievedEvent: Esta clase sirve como un contenedor de los datos que se reciben del cliente. Los datos son almacenados como Strings.
- Socket: En esta clase se tiene la lógica del servidor. En un puerto determinado se encarga de “escuchar” por conexiones entrantes. Por cada conexión entrante se dispara un evento ClientConnectedEvent, y por cada dato recibido del cliente se dispara un evento DataRecievedEvent.
- SocketListener: Es una interfaz que provee métodos de comunicación entre el socket y los objetos que esperan por datos.

Capa de Utilidades:

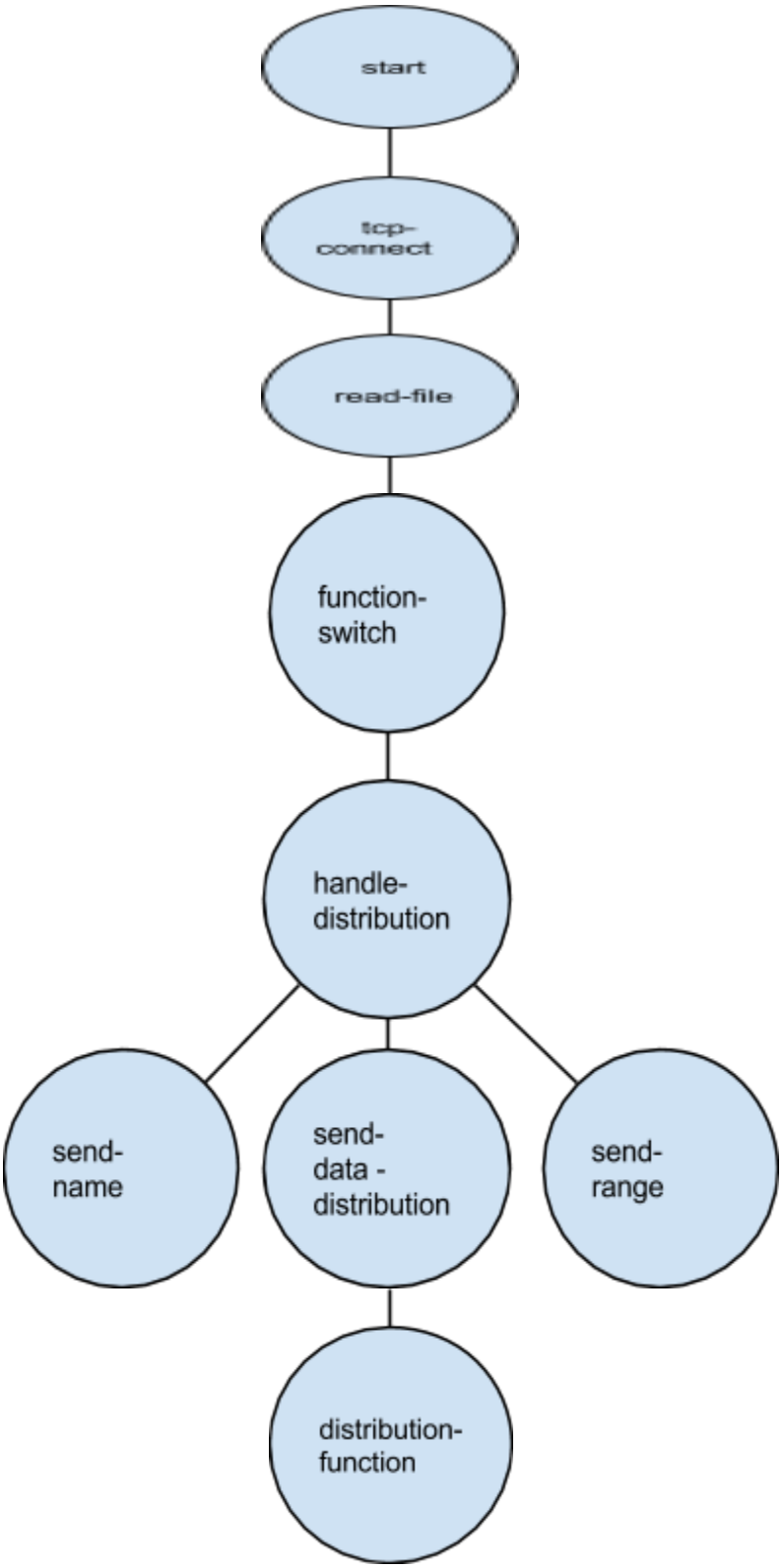
- Utilities: Proporciona métodos útiles para el sistema.

Capa de Interfaz Gráfica:

Para la implementación de la interfaz se utilizó la herramienta JFreeChart, que proporciona clases especiales para el diseño de gráficos.

- DistributionGraph: Es la clase que se encarga de iniciar el programa. Se encarga de crear los objetos necesarios para administrar los datos del gráfico.
- UISocketHandler: Implementa la interfaz SocketListener mencionada anteriormente. Esta clase se encarga de actualizar los datos datos en la interfaz con los datos recibidos por el socket.

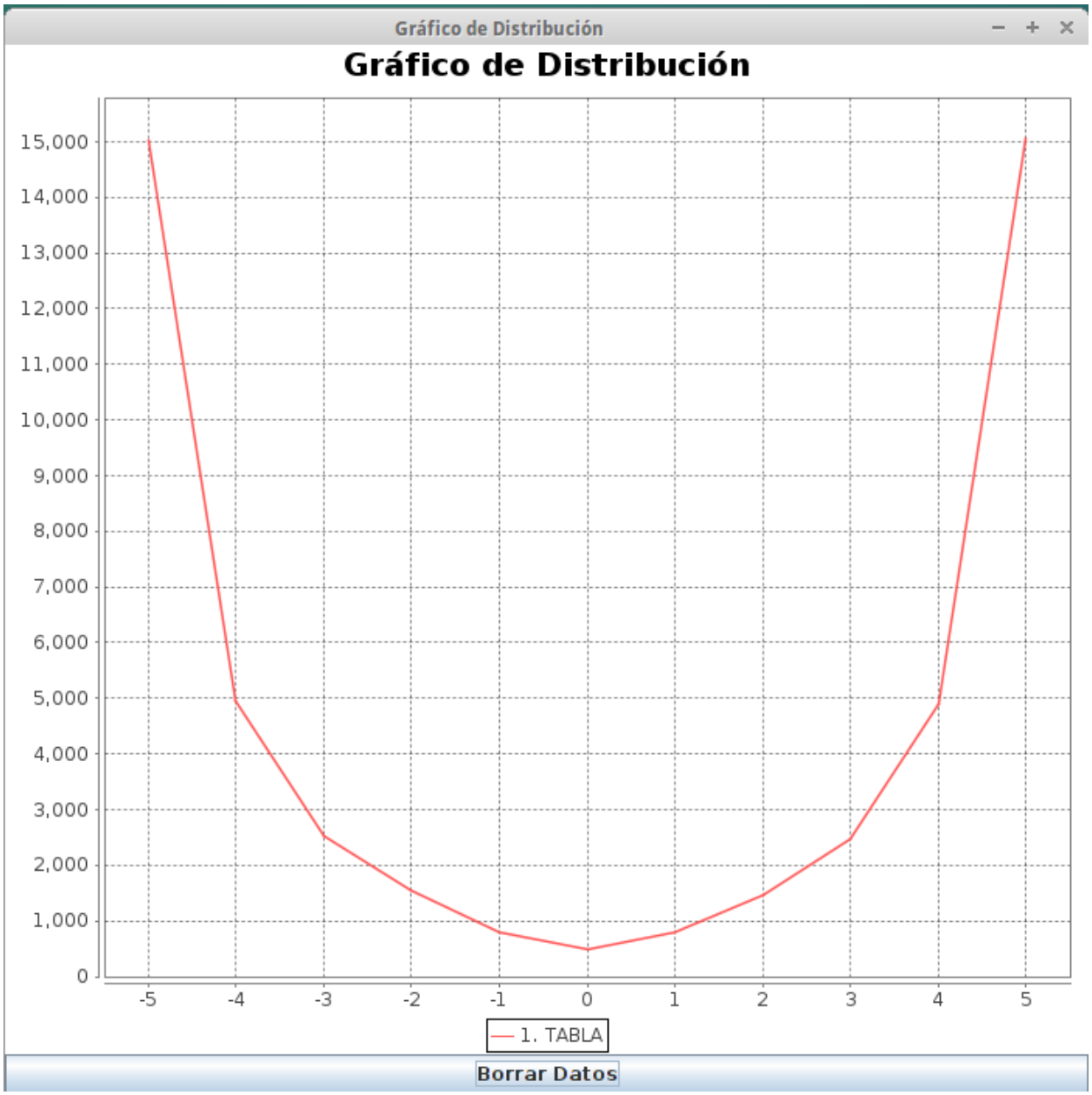
3. Árbol de Invocación de Funciones



4. Muestras de Resultados

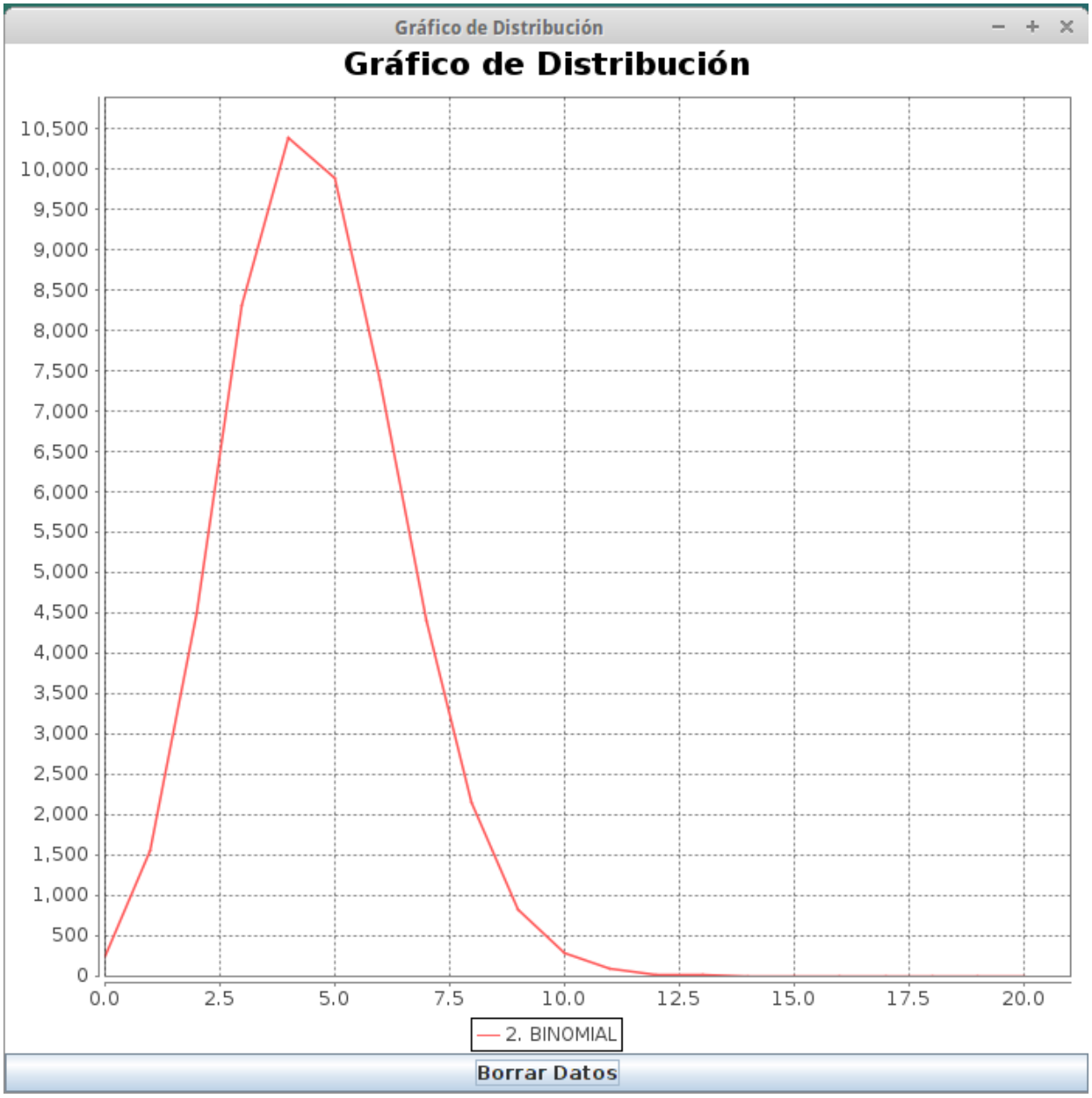
Tabla de Probabilidades

[(-5 0.3) (-4 0.1) (-3 0.05) (-2 0.03) (-1 0.015) (0 0.01) (1 0.015) (2 0.03) (3 0.05) (4 0.1) (5 0.3)]



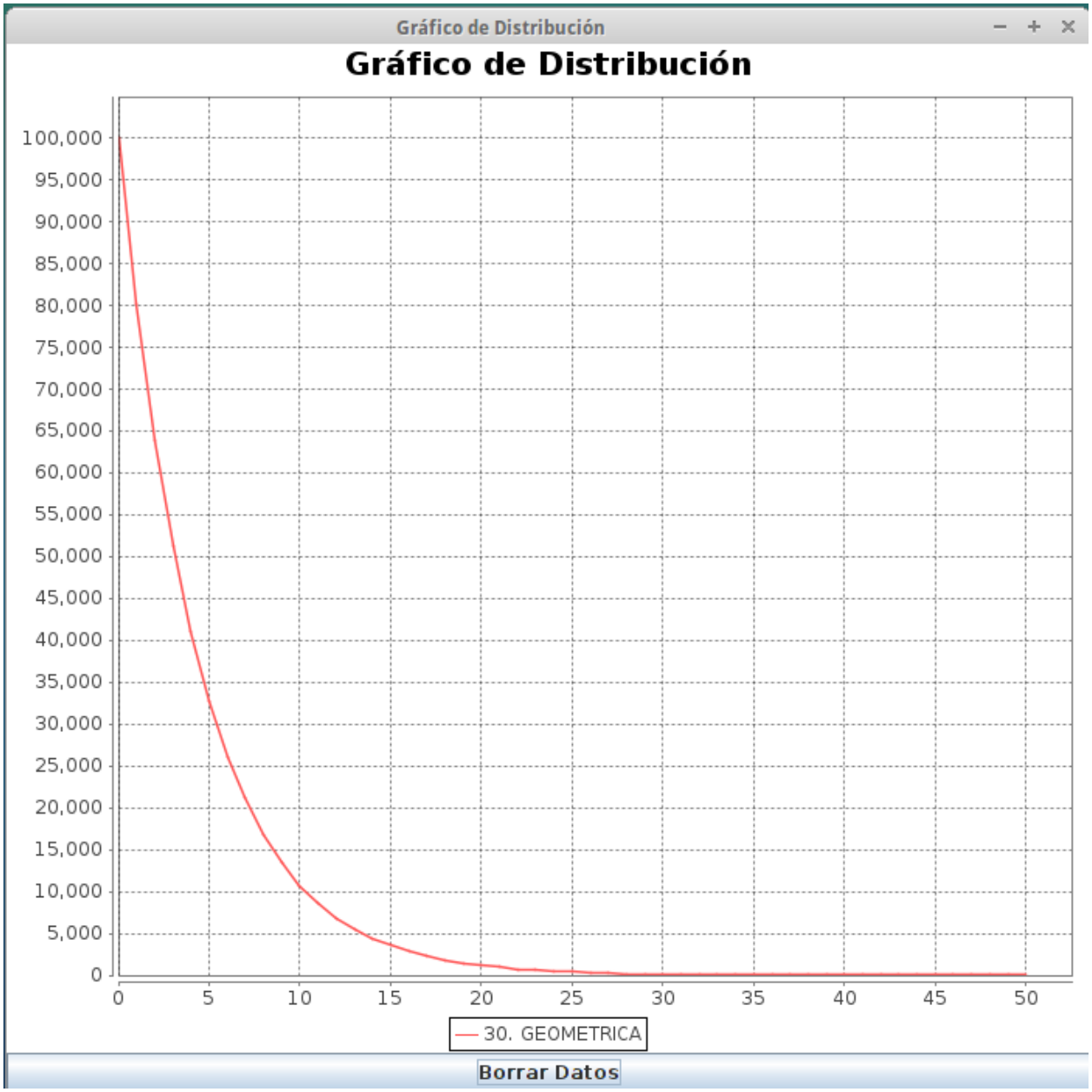
Distribución Binomial

n=20 p=0.23



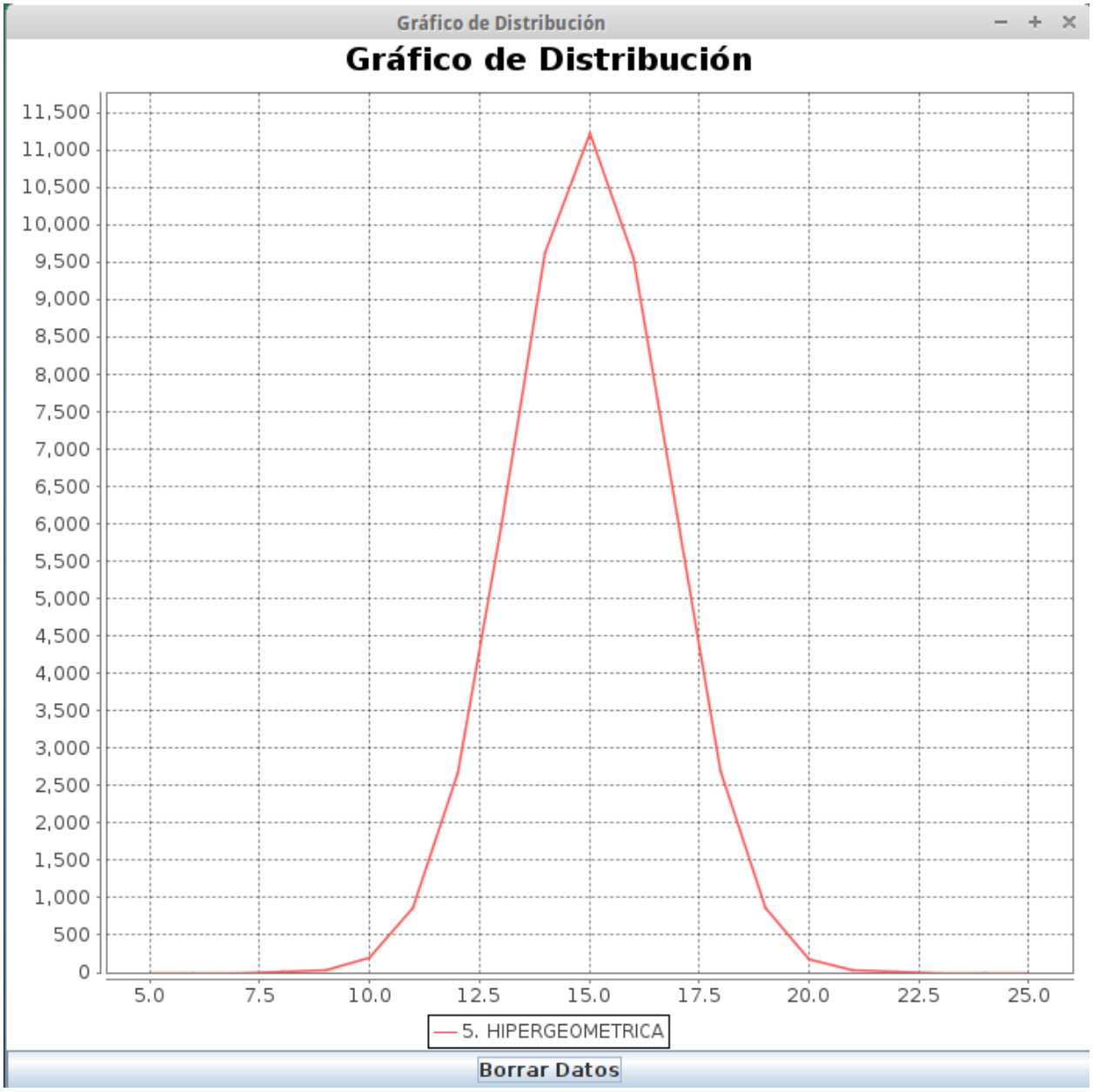
Distribución Geométrica

p = 0.2



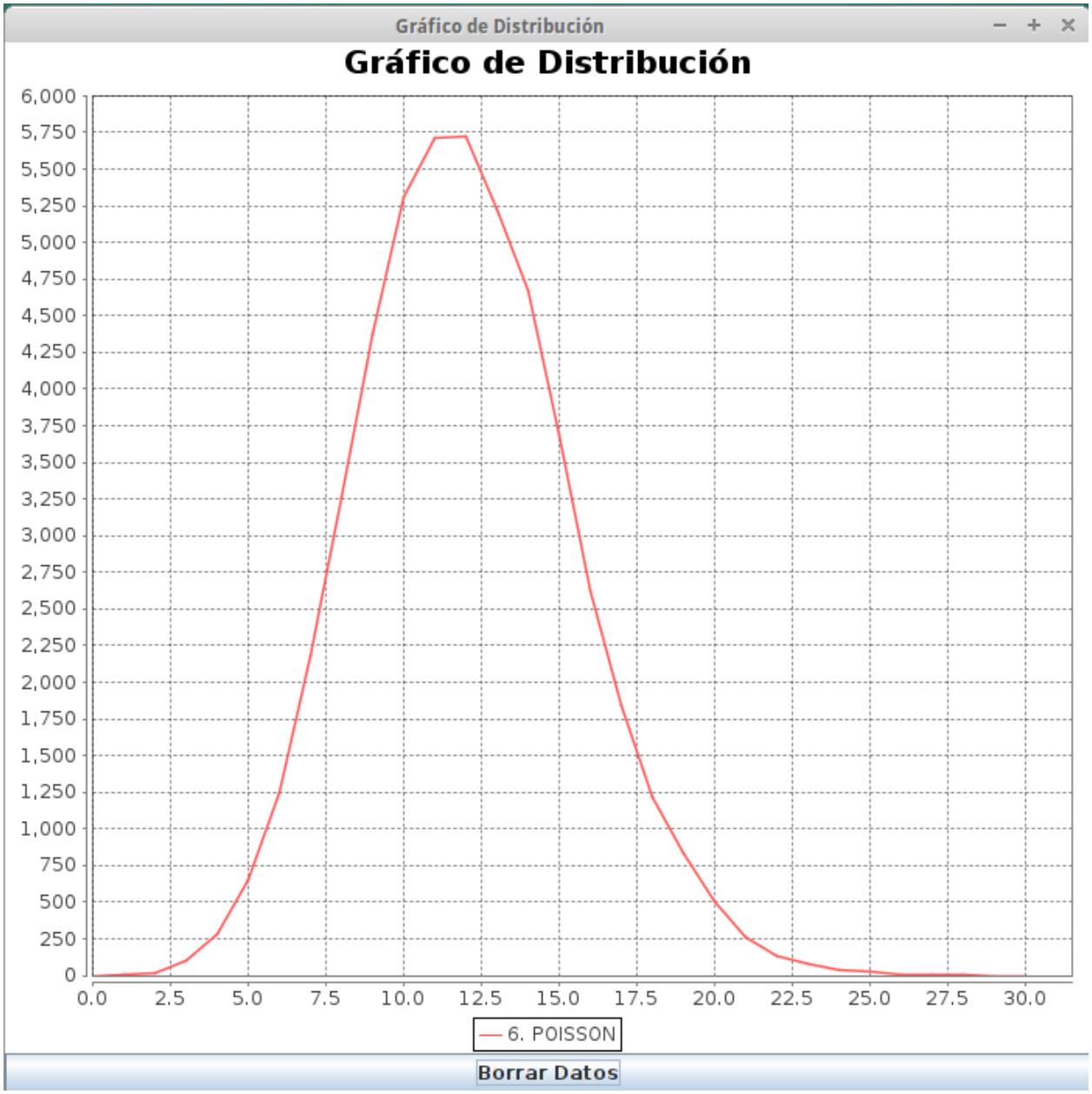
Distribución Hipergeométrica

N=50 d=30 n=25



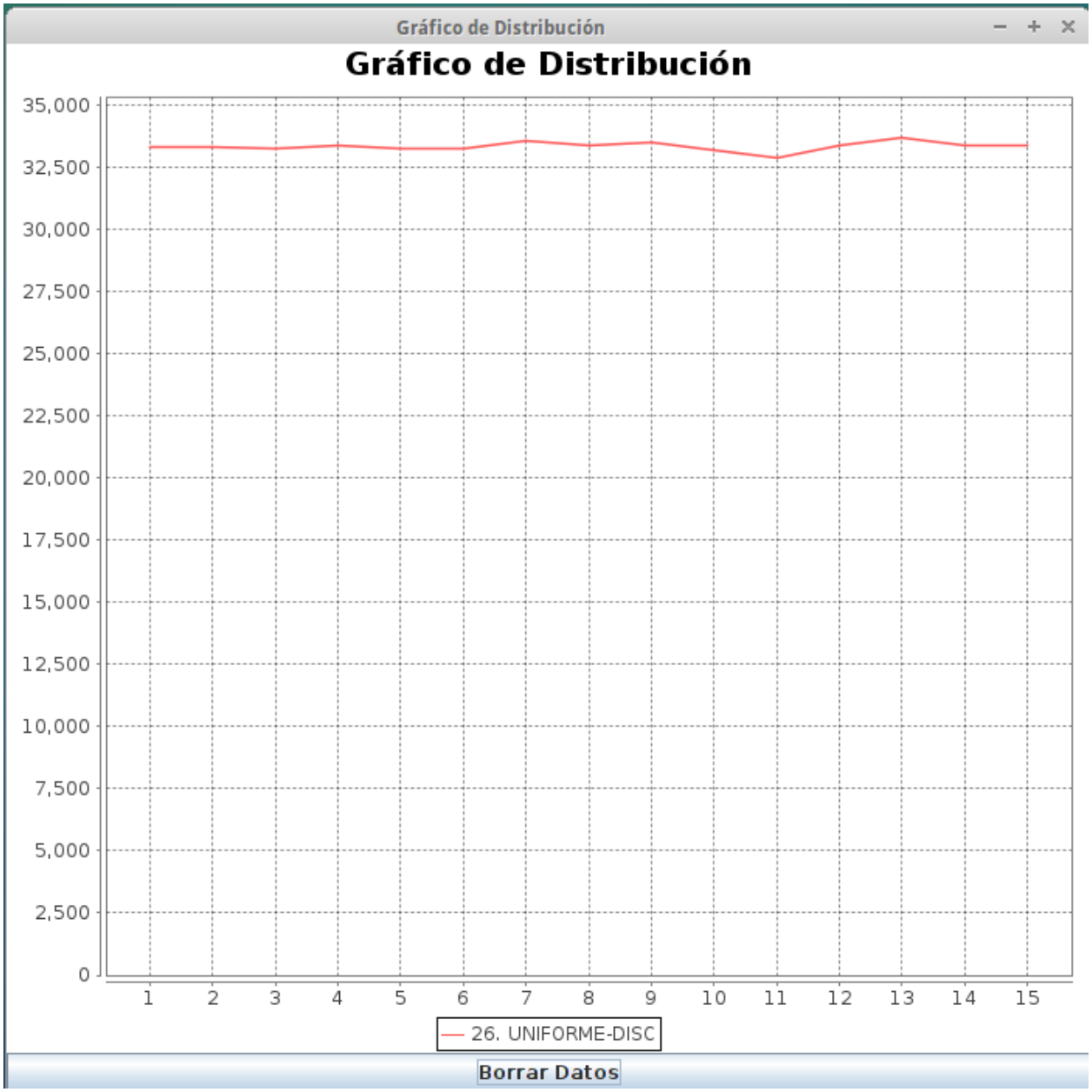
Distribución de Poisson

λ = 12



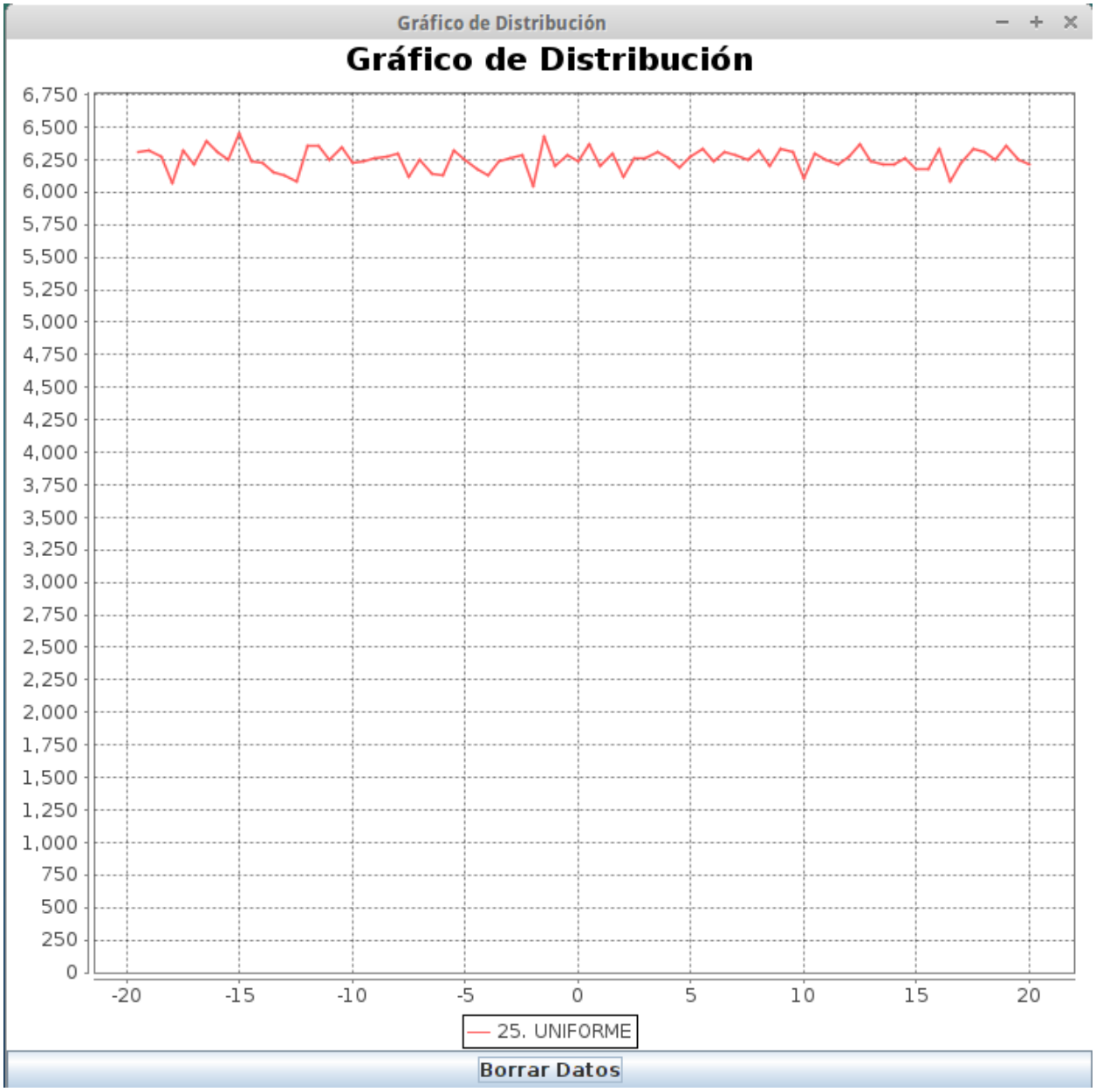
Distribución Uniforme Discreta

Muestra: (1 2 3 4 5 6 7 8 9 10 11 12 13 14 15)



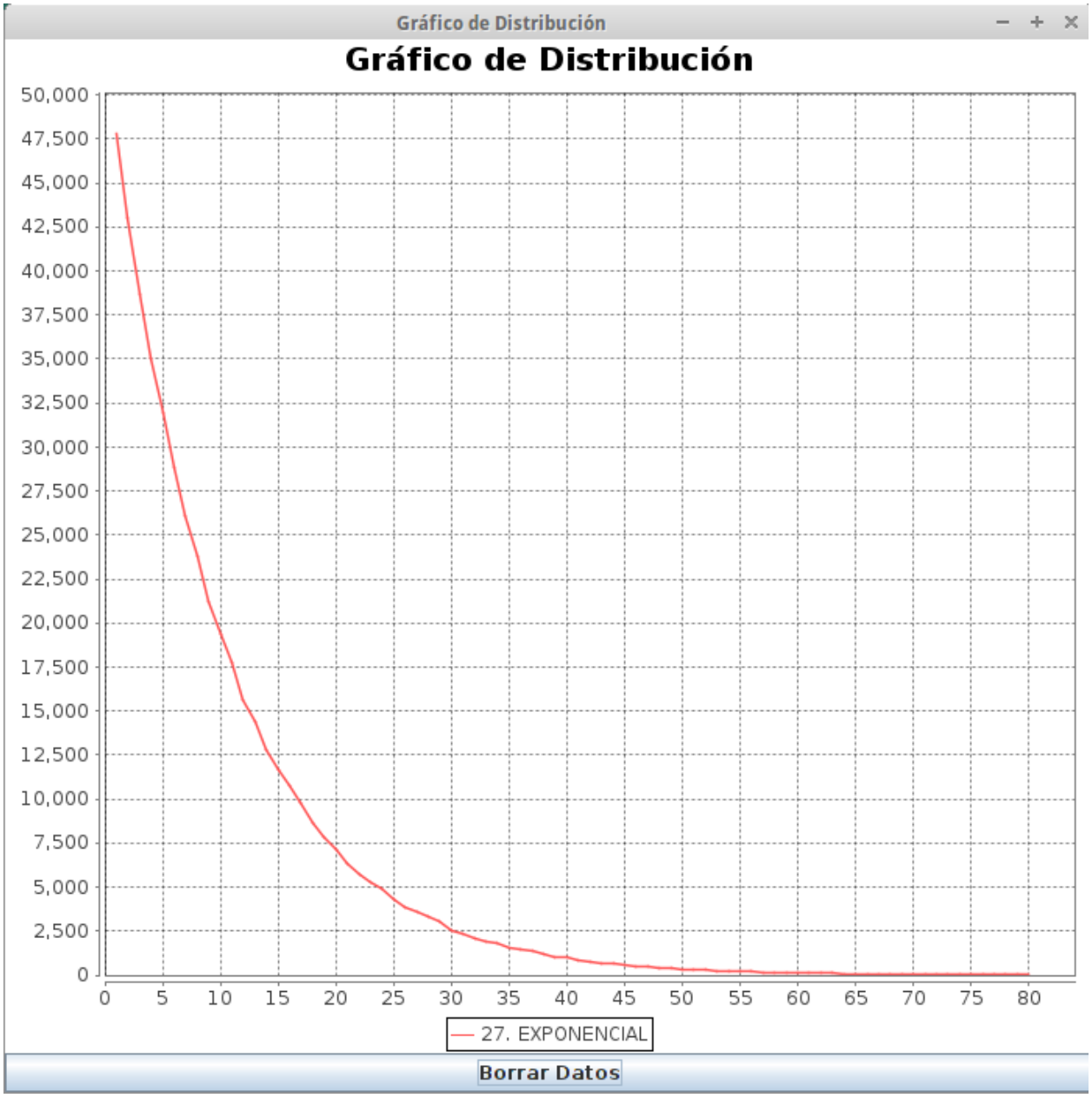
Distribución Uniforme Continua

Rango: [-20, 20]



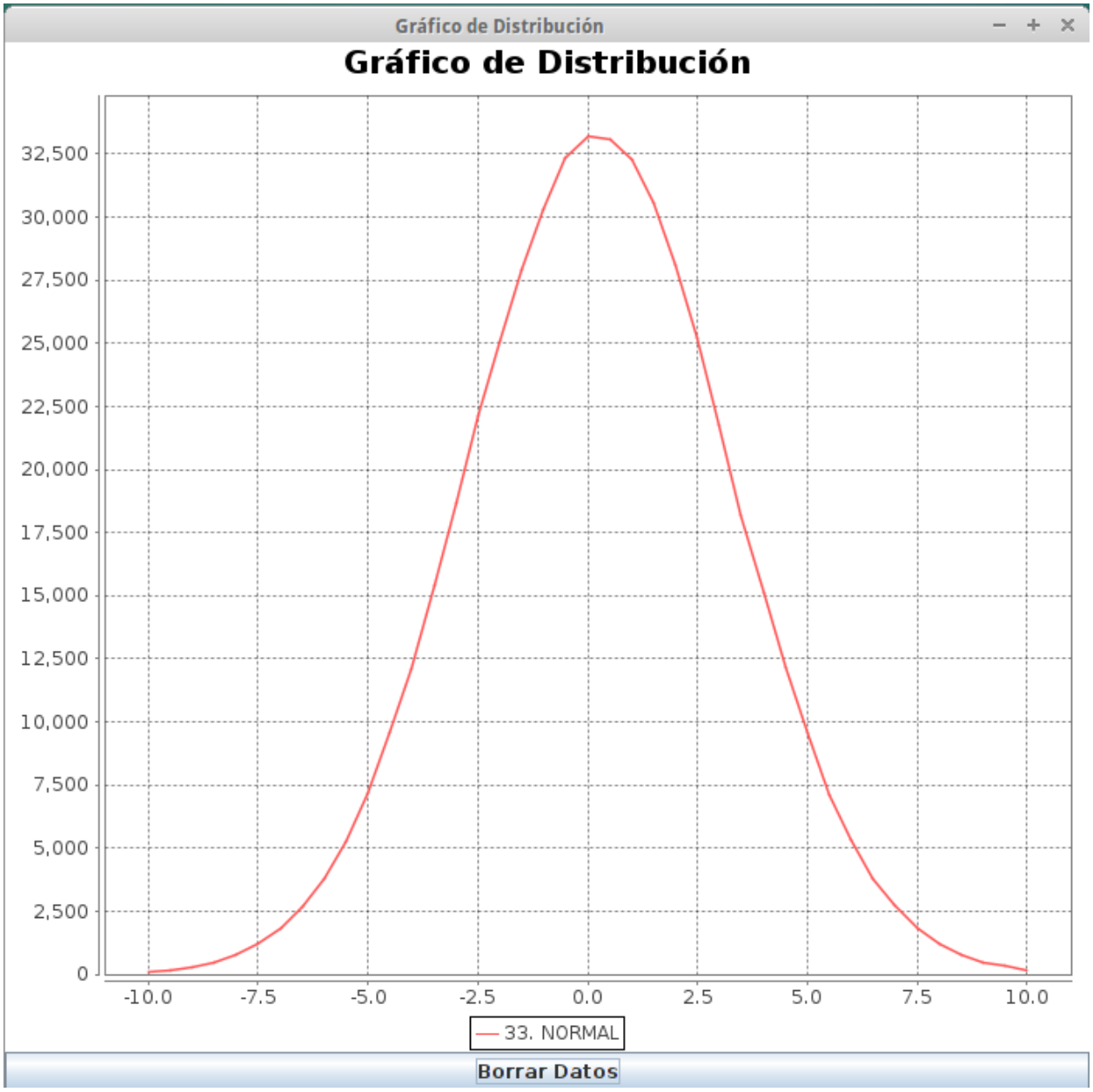
Distribución Exponencial

μ = 10



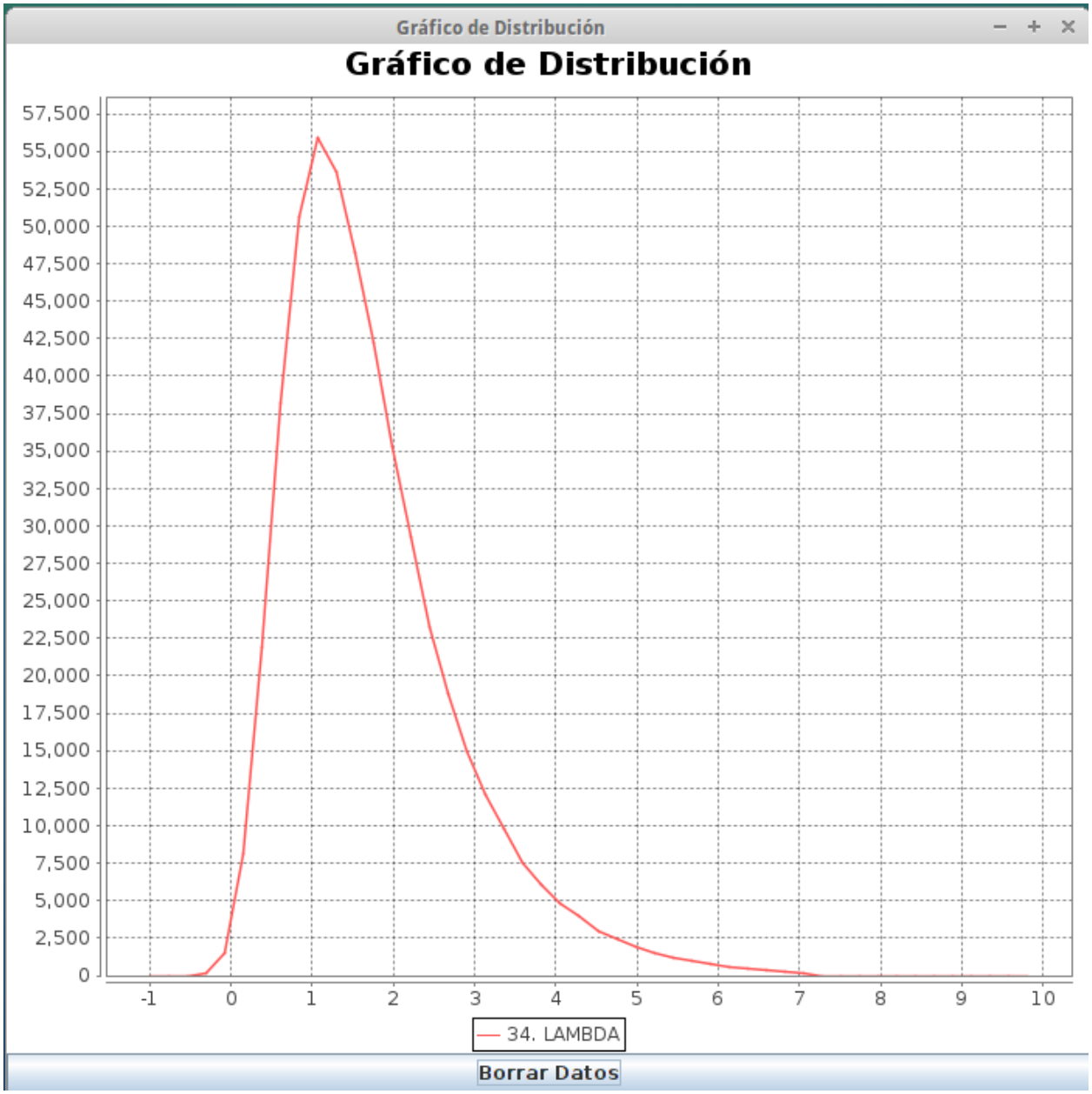
Distribución Normal

μ=0 σ=3



Distribución Personalizada

Moyal μ=1 σ=0.5



5. Texto de Archivos Utilizados en Pruebas

Los archivos utilizados para las pruebas son los siguientes:

- **Tabla de Probabilidades**

```
50000
discreta
(0 0 1)
(tabla ((-5 0.3) (-4 0.1) (-3 0.05) (-2 0.03) (-1 0.015) (0 0.01) (1 0.015) (2
0.03) (3 0.05) (4 0.1) (5 0.3)))
```

- **Distribución Binomial**

```
50000
discreta
(0 20)
(binomial 20 0.23)
```

- **Distribución Geométrica**

```
50000
discreta
(0 50)
(geometrica 0.2)
```

- **Distribución Hipergeométrica**

```
50000
discreta
(5 25)
(hipergeometrica 50 30 25)
```

- **Distribución de Poisson**

```
50000
discreta
(0 30)
(poisson 12)
```

- **Distribución Uniforme Discreta**

```
50000
uniforme
(0 0 1)
(uniforme-disc '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))
```

- **Distribución Uniforme Continua**

```
50000
uniforme
(-20 20 0.5)
(uniforme -20 20)
```

- **Distribución Exponencial**

```
50000
continua
(1 80 1)
(exponencial 1/10)
```

- **Distribución Normal**

```
150000
continua
(-10 10 0.5)
(normal 0 3)
```

- **Distribución Personalizada**

```
50000
continua
(-1 10 0.23)
(lambda (x)
  (/ (exp (- (- 1 x) (/ (exp (/ (- 1 x) 0.5)) 2))) (* (sqrt (* 2 PI)) 0.5)))
```

6. Manual de Usuario Scheme

Para la demostración se usará DrRacket como entorno de desarrollo integrado.

Pasos para ejecutar el módulo de scheme:

1. Debe ir a la parte inferior del código, y llamar a la función “start” indicando la ruta del archivo que se quiere leer, el host del servidor y el puerto por el cuál se va conectar al socket. Se pueden ejecutar varios archivos a la misma vez. El archivo debe estar ubicado en la misma carpeta donde se encuentra el código de scheme.

```
;(start "files/tabla.txt" "localhost" 2020)
(start "files/binomial.txt" "localhost" 2020)
;(start "files/geometrica.txt" "localhost" 2020)
(start "files/hipergeometrica.txt" "localhost" 2020)
(start "files/poisson.txt" "localhost" 2020)
;(start "files/uniforme-disc.txt" "localhost" 2020)
;(start "files/uniforme.txt" "localhost" 2020)
;(start "files/exponencial.txt" "localhost" 2020)
;(start "files/normal.txt" "localhost" 2020)
;(start "files/lambda.txt" "localhost" 2020)
```

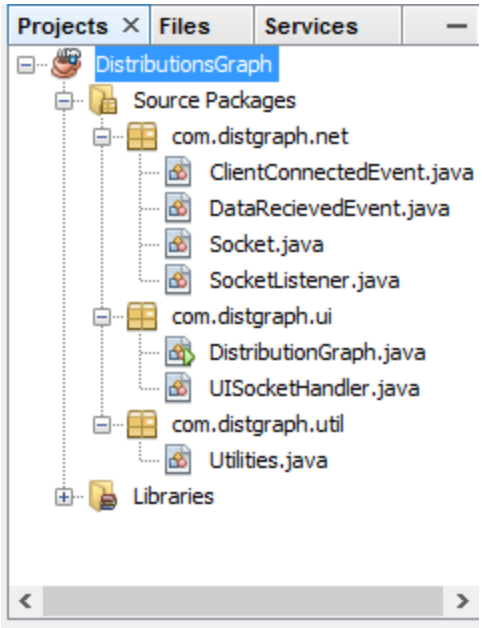
2. Debe presionar el botón “Run”, ubicado en la parte superior derecha de la barra de herramientas. El servidor debe estar escuchando conexiones.

7. Manual de Usuario Java

Para la demostración se usará NetBeans como entorno de desarrollo integrado.

Pasos para ejecutar el módulo de Java:

1. Debe abrir el proyecto desde la ubicación donde se guardó.



2. Debe presionar el botón “Run Project”, triángulo verde, ubicado en la barra de herramientas.