



**Trabajo de Fin de Máster e Iniciación a la Investigación**
**Máster Interuniversitario de Astrofísica UAM-UCM**

# Novel analysis of radial-velocity curves for the detection of super-earths and exo-earths

**Junio de 2013**

**Alumno:**
**Carlos Alberto Gómez González**
**Directores:**
**Luis Dinis y José A. Caballero**

# Abstract

Context. The radial velocity technique is nowadays the most succesful method of detecting exo-planets, especially of the lowest masses. It is however affected by the noise of different sources: magnetic activity, granulation, spots, Poisson error. These noises can be either white (uncorrelated) or red (correlated).

Aims. Our goals are the preparation of a novel methodology of dealing with radial velocity curves containing red noise, and the development of a computer program for automatizing this methodology.

Methods. We have implemented such suite of computer algorithms written mostly in C, and used some MATLAB functions. The methodology proposed is about modelling synthetic radial velocity curves with noise of different types, treating red noise as a first order autorregresive process AR(1), its disentangling from the orginal signal and the fit of the red noise spectrum parameters for generating a test of significance against red noise null hypothesis.

Results. We applied such methodology to investigate the radial velocity curves of a number of synthetic planetary systems with different orbital and noise parameters that resemble real systems: hot Jupiters, super-earths and exo-earths around solar-like stars. As an example, we have calculated what is the maximum red noise in a radial velocity curve of a solar-like star for determining a positive detection of an exo-earth with 99 % confidence level.

Conclusions. The methodology and the program developed for this purpose allowed us to calculate that the maximum red noise amplitude in a radial velocity signal is $\sim$ 1 cm/s for a 99 % confidence exo-earth detection.

**Keywords:** Stars: planetary systems — Techniques: radial velocities — Methods: data analysis

# Índice

# Índice de figuras

# Índice de Tablas

# Capítulo 1

# Introducción

## 1.1. Exoplanetas

La búsqueda de exoplanetas constituye uno de los retos más apasionantes de la Astronomía actual, pues trata de dar respuesta a preguntas que se viene planteando la humanidad desde hace siglos. Ya hace más de dos milenios, en el siglo IV a. C., que Epicuro planteó que "hay infinitos mundos, parecidos al nuestro y diferentes" sentando con esto las bases de una primitiva filosofía racionalista en la que todo se veía como resultado de las leyes de la naturaleza. Varios siglos después, hace unos 500 años, el filósofo italiano Giordano Bruno discutió la posibilidad de planetas alrededor de otras estrellas y presentó su idea de que "existen incontables soles e incontables tierras rotando alrededor de sus soles". Desafortunadamente para él, el planteamiento de esta idea (entre otras ideas revolucionarias para su época) le costaría la condena a la hoguera.

Desde aquel entonces la Astronomía ha progresado enormemente y hoy resulta evidente que nuestro Sol y Sistema Solar no son únicos. Durante varios siglos muchos astrónomos han tratado de detectar exoplanetas, pero esto ha sido posible sólo hasta hace un par de décadas, gracias a los avances en las tecnologías de observación y detección. En 1992 se descubrirían los primeros exoplanetas alrededor de una estrella de neutrones, constituyendo la prueba definitiva de que existen planetas fuera de nuestra Sistema Solar.

El siguiente paso sería encontrar planetas alrededor de una estrella como la nuestra, lo que ocurriría unos años después. A finales de 1995, el mismo año en el que se habían detectado las primeras estrellas enanas marrones, los astrónomos del Observatorio de Ginebra, Michel Mayor y Didier Que-

loz, anunciaron el descubrimiento de el planeta 51 Peg b (Mayor & Queloz 1995), alrededor de una estrella de tipo solar, por la técnica de velocidad radial. Mientras que en el Sistema Solar, los planetas gigantes como Saturno y Júpiter se encuentras en órbitas lejanas, y los planetas pequeños como Venus y la Tierra tienen órbitas más pequeñas, muchos sistemas planetarios extrasolares albergan planetas tipo Júpiter o incluso más grandes y en órbitas más pequeñas que la órbita de Mercurio alrededor del Sol, como es el caso de 51 Peg b. Además, estas órbitas son generalmente más elípticas comparándolas con las del Sistema Solar, que son prácticamente circulares. Las inesperadas características de estos exoplanetas ha hecho que los modelos de formación y evolución dinámica sean revisados para adaptarse a estos nuevos tipos de sistemas planetarios. Al día de hoy 21 de junio del 2013 según `http://exoplanet.eu/` (una de las bases de datos actuales más grandes en exoplanetas) se han descubierto 892 planetas.

Las llamadas supertierras son exoplanetas con masas desde unas pocas masas terrestres hasta un poco menos que la de Urano (1–10 $M_\oplus$). Estos objetos conforman una nueva clase de cuerpos planetarios con características físicas y dinámicas diferentes a los de los planetas terrestres, pero aún así relevantes para las teorías de formación planetaria y habitabilidad. La primera supertierra fue descubierta por Rivera et al. (2005) llamada GJ 876d, y detectada también por el método de velocidad radial. La mayoría de las supertierras presentan semiejes menores a 0.2 UA y sus excentricidades varían de 0 a 0.4. Sus tamaños y masas hacen que sean más fáciles de detectar en comparación a planetas tipo Tierra. También sugieren que las supertierras pueden poseer interiores dinámicos y ser capaces de desarrollar y mantener moderadas atmósferas, lo que las hace potencialmente habitables si sus órbitas caen dentro de la zona de habitabilidad de su estrella madre. Las exotierras son exoplanetas con masa similar a la terrestre o menor y con una composición rocosa.

En el 2011 la misión del Observatorio Espacial *Kepler* publicó una lista de 1235 candidatos a exoplanetas, con 68 candidatos tipo exotierra ($R_p <$ 1.25 $R_\oplus$) y 288 candidatos tipo supertierra ($1.25R_\oplus < R_p < 2R_\oplus$). Hoy en día ya es un suceso común que se se anuncie una nueva supertierra en la zona habitable de su sistema. Ejemplos de ellas en la Fig. 1.1. Según estimaciones hechas este año por los astrónomos del Harvard-Smithsonian Center for Astrophysics (CfA), deben haber por lo menos 17 mil millones de exotierras en la Vía Láctea.

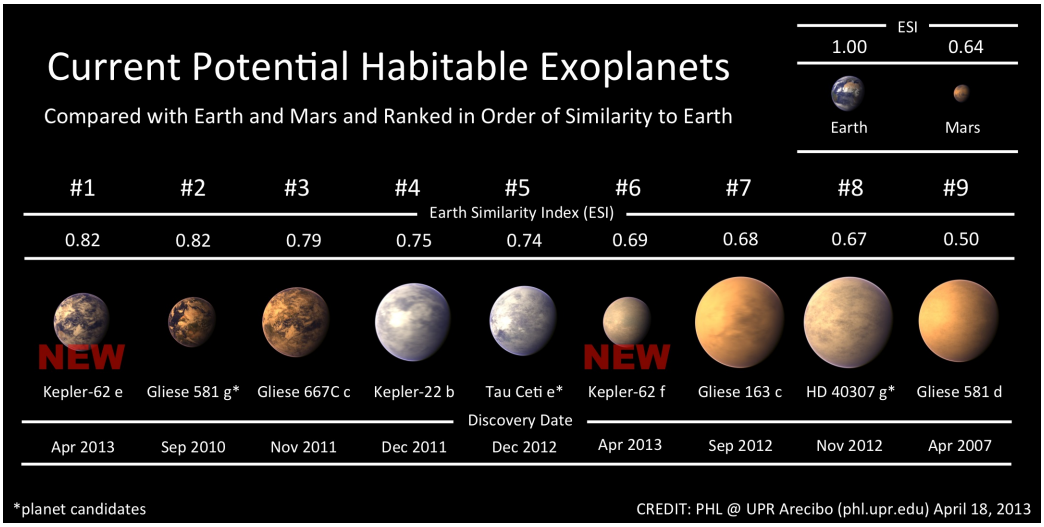



Figura 1.1: Actuales candidatos a planetas potencialmente habitables al 18 de abril de este año, tomado del sitio web del Planet Habitability Laboratory (Universidad de Puerto Rico at Arecibo).

## 1.2. Métodos de detección

Los exoplanetas hasta hace poco no podían ser "vistos", y sólo se podían detectar a través de métodos indirectos, como el de la velocidad radial. Es por eso que hoy la mayoría de detecciones (más del 90 %) han sido hechas por los métodos de velocidad radial y tránsitos.

### 1.2.1. Imagen directa

Es el método más novedoso de todos y con más futuro. Actualmente ya se cuenta una docena de exoplanetas detectados por imagen directa. Este método supone unos retos observacionales inmensos teniendo en cuenta que los planetas son una fuente de luz extremadamente débil en comparación con las estrellas, lo que reduce inmensamente el contraste. Se favorece la obtención de imágenes en circunstancias especiales como cuando el planeta es mucho mayor que Júpiter, está bastante separado de la estrella o está lo suficientemente caliente para emitir luz infrarroja. Además se usan coronógrafos para bloquear la luz de la estrella y aumentar el contraste de la luz de los planetas, ver Fig. 1.2. Si bien no se pueden determinar por este método ni la masa ni el tamaño de los planetas, sus órbitas pueden ser trazadas con precisión y sus espectros pueden ser tomados separadamente de la estrella para obtener

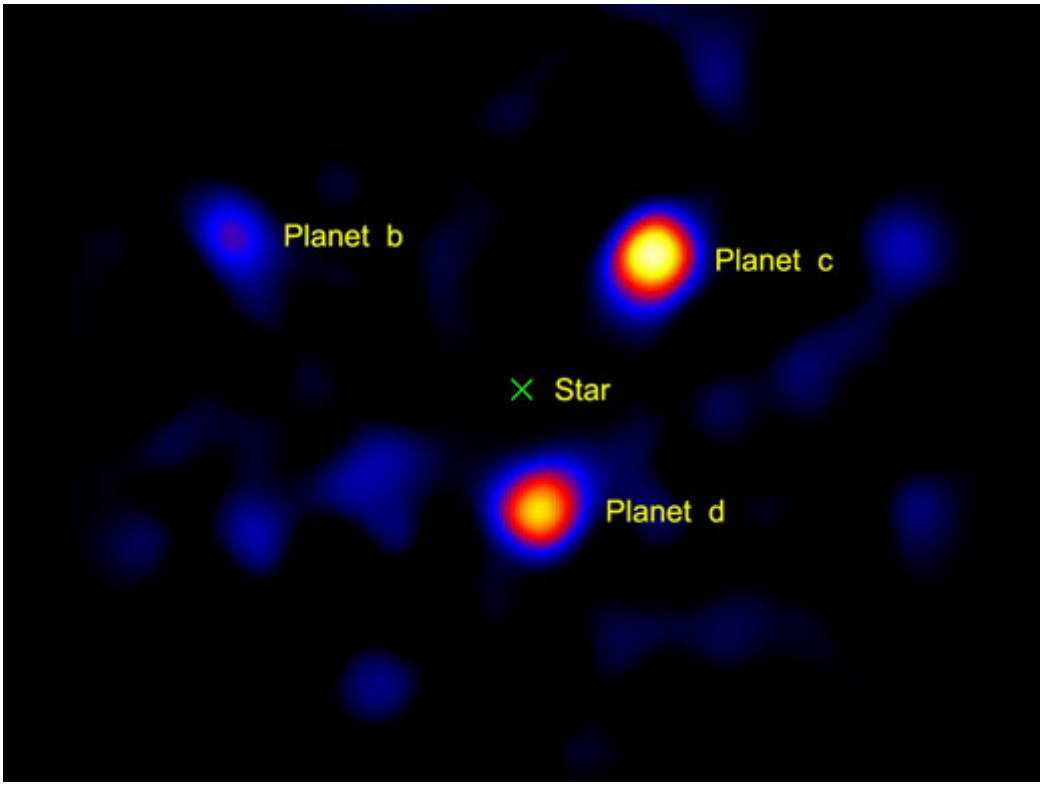



Figura 1.2: Tres planetas alrededor de HR 8799, en la X verde. Se estima que son tipo Júpiter y están a 24, 38 y 68 UA de su estrella. Crédito de la imagen: NASA/JPL-Caltech/Palomar Observatory

composiciones químicas.

### 1.2.2. Tránsitos fotométricos

Este método muestra gran sensibilidad a grandes planetas en órbitas cercanas. El tránsito permite conocer el radio del planeta pero sólo funciona para aquellos que están alineados con nuestro punto de vista desde la Tierra. Lo que se hace es escanear muchas estrellas en amplias zonas del cielo. Hay actualmente numerosos surveys de búsqueda de planetas tipo Tierra y supertierras usando telescopios terrestres y espaciales. Entre estos está el `MEarth` project, un conjunto de ocho telescopios de 40 cm robóticamente controlados en el F. L. Whipple Observatory (Mt. Hopkins, Arizona), que ha sido capaz de detectar la primera supertierra transitando una estrella M (GJ 1214b, Charbonneau 2009). En el espacio, *CoRoT* y *Kepler* por su parte

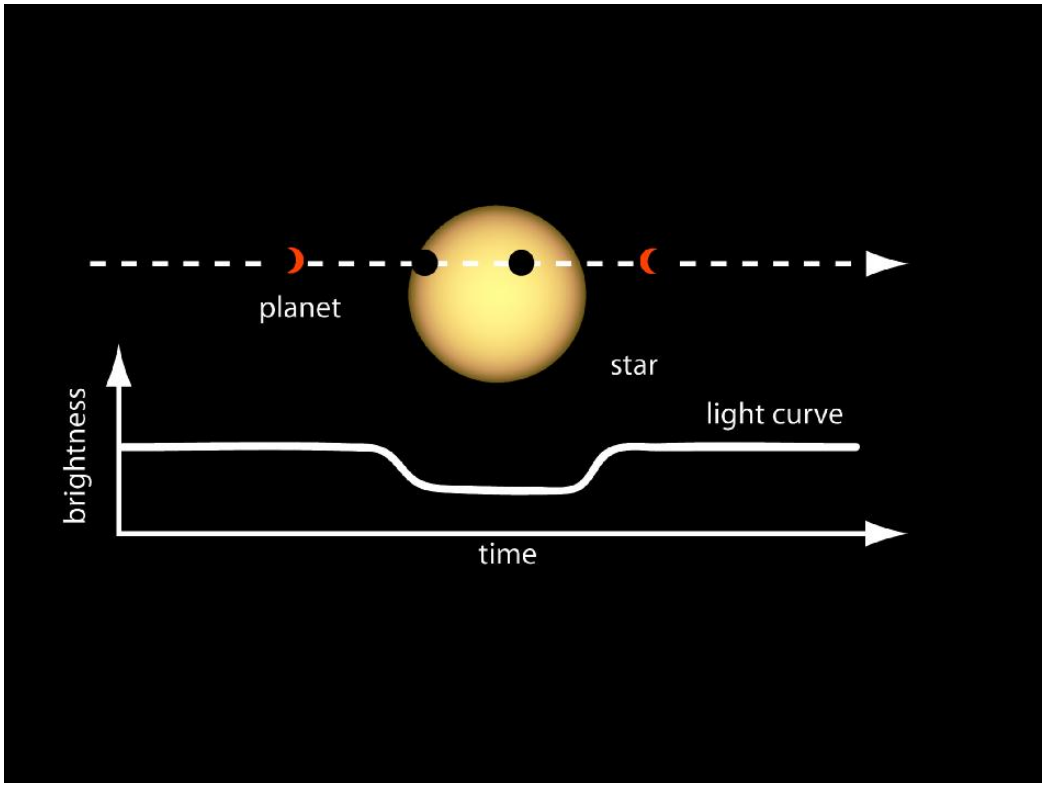



Figura 1.3: Esquema del tránsito de un planeta y el efecto en la curva de luz de su estrella

han sido exitosos detectando numerosas supertierras. Sin embargo este método presenta un alto índice de falsos positivos, y las detecciones deben ser confirmadas por otros métodos, usualmente por velocidad radial.

### 1.2.3. Variación del tiempo del tránsito

El método de variación del tiempo del transito (del inglés "transit timing variation", TTV) consiste en medir el efecto de planetas no detectados cercanos a la órbita de un planeta en tránsito. Estos planetas causan variación en el tiempo y la duración de los tránsitos. Ejemplos de estas variaciones se observan en los tiempos de transito de los planetas terrestres en nuestro Sistema Solar. Mediciones de estas variaciones son usadas para inferir la existencia de un planeta "perturbador" (Steffen & Agol 2005). La señal de esta variación se incrementa cuando los planetas se mueven en resonancia. *Kepler* ha sido capaz de detectar numerosas supertierras por este método.

### 1.2.4. Microlentes

La teoría general de la relatividad explica que cualquier objeto masivo puede actuar como una lente, curvando la luz de un objeto del fondo y aumentando temporalmente su brillo. En este caso la microlente la conforma una estrella que amplifica la luz de una estrella más distante (y su planeta). Este método a diferencia de los anteriores es capaz de detectar planetas a mayores distancias, y es reconocido como el más promisorio para la búsqueda de planetas de baja masa más allá de la "línea de nieve", así como de supertierras que residen lejos de sus estrellas. Es una técnica exitosa en detectar a los llamados "planetas flotadores libres" (free-floating planets), muchos de los cuales son del tipo terrestre o supertierrras. Una gran desventaja de este método es que el chance de repetir el alineamiento es nulo y por tanto una detección no se puede repetir. El *Wide-Field Infrared Survey Telescope* (*WFRIST*), una futura misión de la NASA, usará el método de microlentes para la búsqueda de exoplanetas. Actualmente hay siete diferentes proyectos activos que usan esta técnica, entres estos *MicroFUN* (*Microlensing Follow-Up Network*), *MOA* (*Microlensing Observations in Astrophysics*), *PLANET* (*Probing Lensing Anomalies NETwork*), han sido exitosos en la detección de numerosos planetas.

### 1.2.5. Astometría

Este método consiste en la medición precisa de la posición de una estrella en cielo y el cambio que experimenta con el tiempo. Es el método más antiguo de búsqueda de exoplanetas y es popular por su éxito en la caracterización de estrellas binarias astrométricas. Sólo hasta 1992 el *Telescopio Espacial Hubble* consiguió caracterizar un sistema ya descubierto (Gliese 876) por el método astrométrico. Por el momento no se ha confirmado ningún exoplaneta por este método pero en el futuro con la misión espacial *Gaia* tal vez se consiga más éxito. Una ventaja potencial de este método es que es más sensible para órbitas grandes, lo que complementaría otros métodos como el tránsito fotométrico. Las observaciones deberán llevarse a cabe por largos períodos (años) para completar una órbita.

### 1.2.6. Velocidad radial

La velocidad radial es aquella con la que se mueve una estrella en la dirección radial desde el baricentro del sistema extrasolar; esta puede ser determinada a partir del desplazamiento en las líneas del espectro por el efecto Doppler. La espectroscopia Doppler mide la influencia gravitatoria de

un planeta sobre su estrella a través de medidas de la luz de esta última. Es un método indirecto por tanto. El método de medición de la velocidad radial consiste en observar una estrella y en seguida un espectro de comparación (de la luz de una lámpara) que mostrará las líneas espectrales sin corrimiento, o en reposo para el observador. Esto se repite varias veces en el transcurso de un periodo de tiempo determinado lo que da una serie temporal con los valores de la velocidad radial observados (la curva de velocidad radial).

En los 50s teníamos una precisión de 1 km/s y hoy se tiene incluso menos de 1 m/s de precisión en la medición del corrimiento Doppler. A pesar de que la sensibilidad actual de este método permite hoy en día la detección de supertierras alrededor de estrellas tipo Sol, las estrellas de baja masa como las enanas tipo M son los objetivos más promisorios para la búsqueda de planetas tipo Tierra y supertierras. Esto se debe principalmente al hecho de que estas estrellas presentan mayor perturbación debida a su compañero planetario, y a que, como sugieren las simulaciones de formación planetaria, los planetas alrededor de enanas M son generalmente menores a las gigantes gaseosos y más probablemente en órbitas cercanas. Existen varios surveys enfocados en la búsqueda de supertierras alrededor de estrellas tipo M, como el *Lick-Carnegie Exoplanet Survey*, el *M2K Planet Search Project* y el *HARPS Search for Southern Extrasolar Planets*, con bastante éxito. En 2014 se pondrá en marcha el proyecto CARMENES (Quirrenbach et al. 2012,`http://carmenes.caha.es`) que buscará planetas terrestres en torno a estrellas de baja masa desde el Observatorio de Calar Alto. La misión espacial *Gaia* espera encontrar gran número de planetas tipo Júpiter por velocidad radial.

## 1.3. Ruido estelar y su impacto en las mediciones de velocidad radial

Más allá de la calibración en longitud de onda y los retos instrumentales, el último factor que limita la precisión de las mediciones de velocidad radial es la variabilidad intrínseca de las estrella y el ruido que esta genera. A continuación se resumen algunas de las fuentes principales de ruido estelar.

### 1.3.1. Oscilaciones de modo $p$

Las oscilaciones de modo $p$ son ondas acústicas en los que la presión actúa como fuerza restauradora (de ahí la $p$ del nombre). Su dinámica está

determinada por la variación en profundidad de la velocidad del sonido en el interior de la estrella. En el Sol las oscilaciones de los modos $p$ tienen frecuencias características mayores de 1 mHz y son especialmente intensas en el rango de los 2–4 mHz lo que se corresponde con las "oscilaciones solares de cinco minutos". En estrellas tipo solar tienen períodos típicos de algunos minutos y amplitudes típicas de algunas decenas de cm/s en velocidad radial (hasta algunos m/s gracias a que estos modos se superponen). En estrellas de baja masa poco evolucionadas este tipo de ruido es menor. En general no es un problema grande, son de pequeña escala temporal y hay técnicas para obviar esta fuente de ruido.

### 1.3.2. Convección, granulación y supergranulación

Se da en estrellas que tienen una cubierta convectiva, debido a movimientos convectivos de gran escala en las capas exteriores. La textura granulada esta compuesta de un gran número de celdas con flujos ascendentes (material caliente que viene desde capas más profundas) y descendentes (material que se ha enfriado en la superficie). Afortunadamente para búsquedas planetarias el gran número de gránulos en la superficie ($\sim 10^6$) promedia de manera efectiva los campos de velocidades. Aún así el efecto debido a la granulación se estima del orden de los m/s para el Sol y probablemente menos para enanas tipo K. La escala de tiempo típica es de diez minutos para el Sol. En escalas de tiempo de algunas horas a un día ocurre un efecto similar llamado supergranulación (estructuras convectivas de mayor escala en la fotosfera) que todavía no se entiende del todo bien. La efectos relacionados a la granulación pueden introducir un ruido a tener en cuenta para precisiones debajo del metro por segundo.

### 1.3.3. Actividad magnética y manchas

Campos magnéticos en la superficie de estrellas tipo Sol y enanas M son responsables por un número de fenómenos fotosféricos que introducen ruido en mediciones de velocidad radial precisas. Entre estos están manchas frías y "playas brillantes" que pueden cubrir una variable fracción del disco estelar, evolucionar en el tiempo y son transportadas por la rotación de la estrella. Representan el verdadero problema en las medidas de velocidad radial.

Estas estructuras afectan las medidas de velocidad radial de diversas formas, por ejemplo, cambiando la forma de las líneas espectrales e introduciendo modulaciones en la velocidad radial medida en escalas de tiempo compa-

rables al período de rotación de la estrella. La actividad estelar en general es un gran problema para búsquedas de exoplanetas alrededor de estrellas jóvenes ($\tau \leq 600$ Ma), con manchas oscuras causantes de variaciones en la velocidad radial mayores a 10–100 m/s. Una posible solución sería observar en el infrarrojo cercano donde el contraste de las manchas oscuras es menor. Un diagnóstico bastante usado es el bisector de la línea, que traza las variaciones de la forma de la línea debida a, por ejemplo, las manchas oscuras. Esta actividad depende principalmente de el área cubierta por las manchas oscuras y las playas, la velocidad rotacional de la estrella y la duración típica de de las regiones activas. Acumular una gran cantidad de puntos (medidas) por ciclo de rotación estelar debe permitir promediar el ruido estelar, aunque los últimos niveles de precisión siguen siendo motivo de controversia.

# Capítulo 2

# Metodología

## 2.1. Análisis de ruido de curvas de velocidad radial

Ahora nos centraremos en el método de velocidades radiales y el impacto de ruido en este. La descripción de la metodología utilizada en el presente trabajo se hará a la par de la descripción del programa `RedNoise` (código fuente en el Apéndice) que fue desarrollado para el presente análisis de ruido en curvas de velocidad radial.

`RedNoise` permite el modelado curvas de velocidad radial de una estrella con un planeta, con la posibilidad de añadir distintos tipos de ruido a la serie temporal. Con una implementación del algoritmo Lomb-Scargle el programa puede encontrar la frecuencia con mayor significación [1] estadística en esta señal (periodo orbital del planeta si se trata de una curva de velocidad radial). Se le presta central atención a la generación del ruido rojo, modelado como un proceso autorregresivo de primer orden AR(1), y a su posterior extracción de la señal. Los parámetros del espectro de este ruido rojo extraído posteriormente son ajustados para calcular un espectro teórico y unos límites de significación con que comparar los picos del periodograma de la señal (curva) de velocidad radial modelada.

El código está implementado en C, con el llamado a algunas funciones (scripts) en MATLAB/Octave. El código está dividido en funciones (módulos) y está extensamente comentado. En su proceso de desarrollo se utiliza

[1] La palabra "significancia" es comúnmente usada en estadística, pero no figura en el Diccionario de la Real Academia Española, por lo que en este texto utilizamos significación.

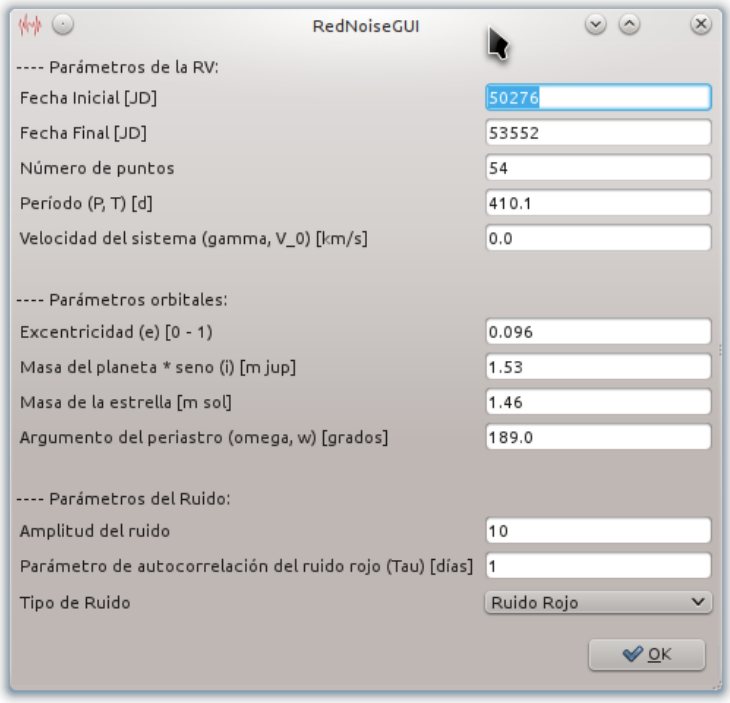



Figura 2.1: Captura de pantalla del GUI o interfaz gráfica de `RedNoise`.

un historial de cambios y un sistema de control de versiones. El desarrollo se ha hecho en un entorno completamente Linux, y no se ha perseguido el uso multiplataforma (aunque el código C se debería compilar sin problemas en otros sistemas operativos donde existan una versión de GCC o GNU Compiler Collection). Se hace uso de la herramienta GNU Make para automatizar el proceso de compilación. Sólo basta con ejecutar en el shell (línea de comandos) el comando `./make` desde el directorio raíz de `RedNoise`.

Una vez compilado el programa puede ser invocado en modo consola ejecutando `./rn`. Una ayuda rápida acerca de los argumentos se obtiene ejecutando `./rn --help`. `RedNoise` posee además un modo gráfico (básico) o GUI (Graphical User Interface) que facilita la introducción de los valores de los diferentes parámetros, una de las ventanas se muestra en la Fig. 2.1. Para invocarlo en modo GUI se puede dar doble click sobre el script `rednoise` o ejecutarlo desde el shell `./rednoise`. La interfaz gráfica está escrita en un script con YAD (Yet Another Dialog Program) que corre en cualquier

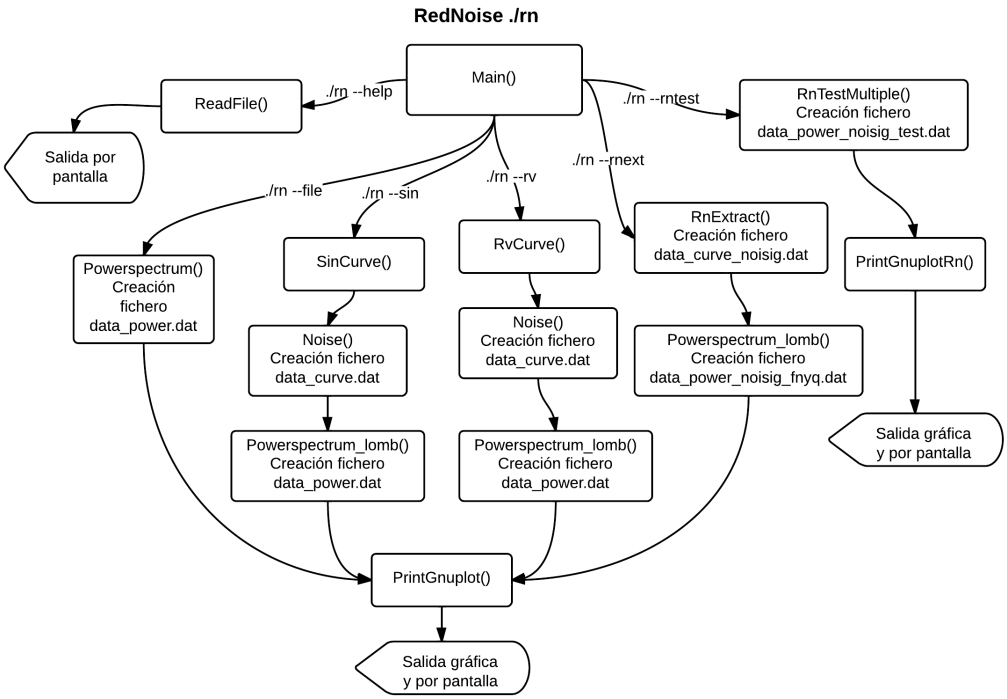



Figura 2.2: Diagrama de bloques que muestra brevemente el flujo de las funciones dentro de `RedNoise`.

distribución UNIX/Linux; este debe ser instalado manualmente ya que no suele venir preinstalado. La salida de gráficos y diagramas se hace a través de `gnuplot` que viene preinstalado en la mayoría de sistemas UNIX/Linux.

En el diagrama de bloques de la Fig. 2.2 se intenta mostrar gráficamente (sin pretender ser riguroso ni completo) las funciones (una en cada bloque) principales de `RedNoise` y el flujo de las opciones principales en modo consola.

Los pasos principales de la metodología propuesta son:

- Generación de una curva de velocidad radial sintética con ruido de diferentes tipos,
- hacer el ajuste de los parámetros orbitales de la curva de velocidad radial, usando el EXOFIT,
- generar una nueva curva de velocidad radial a partir de estos parámetros, esta vez sin ruido,

- extraer el ruido, restando estas señales,
- tomar el espectro de este ruido, y ajustarlo a la función de densidad espectral teórica del proceso AR(1),
- con los parámetros del ajuste, generar el test de significación y evaluar nuestro pico.

### 2.1.1. Generación de curvas sintéticas de velocidad radial

Para el estudio del ruido en las curvas de velocidad radial empezamos por la generación de curvas sintéticas a las cuales posteriormente podremos agregar dicho ruido. Como primera aproximación se tomó la velocidad radial como una curva sinusoidal a la cual se le suma una señal de ruido. La función encargada se llama `SinCurve()`. El tiempo de esta serie se toma aleatorio en un intervalo entre `t_0` (tiempo inicial) y `t_f` (tiempo final). Para la generación de números pseudoaleatorios no se utilizó la función la función predeterminada `rand()` o sus derivados, en vez de esta se utilizó un algoritmo publicado por Marsaglia & Zaman (1987). Este algoritmo es citado como uno de los mejores generadores de números pseudoaleatorios en revisiones de literatura en este tema. En los códigos se encontrará como la función `RandomUniform()`. A continuación un resumen descriptivo de la función `SinCurve()`.

```
/*************************************************************************
 * FUNCIÓN: SinCurve
 *************************************************************************
 * Función para generar una sinusoide como primera aproximación a una RV. Llama
 * a la función de generación de ruido
 *************************************************************************
 * Argumentos:
 * t\_0     tiempo inicial de la serie temporal
 * t\_f     tiempo final de la serie temporal
 * P        periodo orbital del planeta
 * N        número de puntos de la curva
 * K\_1     semiamplitud de la sinusoide
 * gamma    RV\_0, velocidad sistémica o velocidad radial del sistema (centro de
 *          masa)
 * phi      desfase de la sinusoide
 * noise    amplitud del ruido
 * tau      parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *          se usa en el caso de querer generar ruido rojo
 * arg      argumento switch para escoger el tipo de ruido a agregar
 *************************************************************************
 * Devuelve 0 si finaliza correctamente
 *************************************************************************/
```

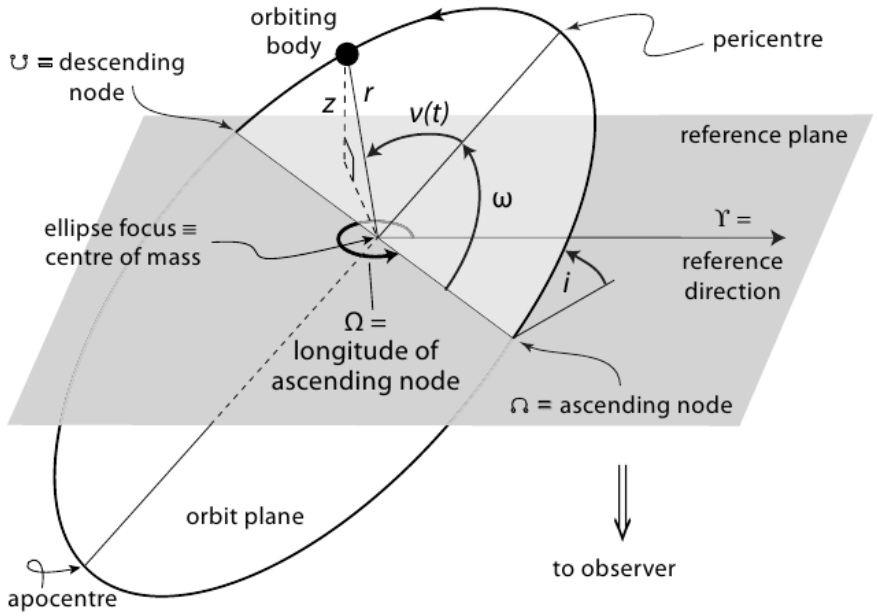



Figura 2.3: Órbita elíptica en el espacio (Tomado de Exoplanet Handbook, Michael Perryman, 2011).

#### 2.1.1.1. Parámetros orbitales de la velocidad radial

El siguiente paso es generar curvas de velocidad radial a partir de los parámetros orbitales. La orientación de la órbita de un exoplaneta en el espacio puede verse en la Fig. 2.3. La velocidad radial de una estrella con un planeta orbitando puede expresarse como una función de la anomalía verdadera y otros parámetros orbitales con la expresión:

$$V_r = V_0 + K_1[\cos(\omega + v) + e\cos(\omega)] \tag{2.1}$$

donde $K_1$ es la semiamplitud de la variación de la velocidad radial, $V_0$ (o $\gamma$) es la velocidad del centro de masas del sistema estrella-planeta con respecto al baricentro del Sistema Solar, $e$ es la excentricidad, $\omega$ el argumento del periastro (el ángulo que va desde el nodo ascendente hasta el periastro medido en el plano orbital del planeta y en sentido de su movimiento) y $v$ la anomalía verdadera (el ángulo entre la dirección del periastro y la posición actual del planeta medida desde el centro de masas). La semiamplitud a su vez puede ser expresada así:

$$K_1 = \frac{2\pi a \sin(i)}{\sqrt{1-e^2}} \tag{2.2}$$

donde $a$ es el semi-eje mayor y $P$ es el periodo. Luego utilizando la tercera ley de Kepler que relaciona el periodo con el semi-eje mayor y que en su forma general es $P^2 = \frac{4\pi^2}{GM}a^3$, obtenemos:

$$K_1 = \left(\frac{2\pi G}{P}\right)^{1/3} \frac{M_2 \sin(i)}{M_1^{2/3}} \frac{1}{\sqrt{1-e^2}} \tag{2.3}$$

Por este método de velocidad radial no se puede determinar el ángulo de inclinación $i$, por lo que en general sólo obtenemos una estimación mínima de la masa del planeta (asumiendo una inclinación de 90 grados). Ahora calculamos la anomalía verdadera $v$ como función del tiempo, para lo que utilizamos:

- la ecuación para la anomalía media $M = \frac{2\pi}{P}(t-T)$ (para el tiempo $(t-T)$ después del paso por el periastro), o el ángulo que forma con el eje de la elipse un planeta ficticio que gira con movimiento uniforme sobre una circunferencia cuyo diámetro coincide con el eje principal de la elipse (llamada circunferencia principal),
- la ecuación de Kepler que relaciona la anomalía media y la anomalía excéntrica, $M = E - e\sin(E)$. Esta se resuelve iterativamente hasta alcanzar una diferencia menor de 0.000001 radianes. Se ha observado que con seis pasos ya hay convergencia, aunque pueden ser muchos más para excentricidades grandes (atípicas para exoplanetas),
- y la relación geométrica entre la anomalía verdadera y la anomalía excéntrica $\cos(v) = \frac{\cos(E)-e}{1-e\cos(E)}$ (el ángulo medido desde el centro de la elipse que forma la proyección del planeta sobre la circunferencia principal de la elipse y el eje de esta).

Como vemos en principio se necesitan seis parámetros orbitales para obtener la velocidad radial:

- $P$ – periodo,
- $e$ – excentricidad,
- $K_1$ – semiamplitud,
- $V_0$ – velocidad radial del sistema,

- $\omega$ – argumento del periastro,
- $T$ – tiempo de paso por el periastro,

En el caso de que hubiese más de un planeta y si consideramos que estos no interaccionan entre sí, podemos sumar las contribuciones keplerianas que cada uno hace por separado, o sea, sumar varias ecuaciones del tipo 2.1. En esta aproximación de primer orden se identifica la señal del planeta dominante y se sustrae de los datos observacionales, repitiendo para todas las señales de planetas presentes.

La función de `RedNoise` que genera las curvas de velocidad radial es `RvCurve()`. Esta utiliza parámetros secundarios como $m_2 \sin(i)$ y $m_1$ (que pueden ser extraídos de la literatura o bases de datos) para calcular $K_1$. El tiempo de igual forma se genera aleatoriamente:

```
t[i] = t_0 + RandomUniform() * (t_f - t_0);
```

A continuación un resumen descriptivo de la función `RvCurve()`:

```
/*****************************************************************************
 * FUNCIÓN: RvCurve
 *****************************************************************************
 * Función para la generación de curvas RV modelando a partir de los parámetros
 * orbitales. Esta función al final llama a la función de generación de ruido
 *****************************************************************************
 * Argumentos:
 * t_0      tiempo inicial de la serie temporal
 * t_f      tiempo final de la serie temporal
 * P        periodo orbital del planeta
 * N        número de puntos de la curva (medidas RV)
 * gamma    RV_0, velocidad sistémica o velocidad radial del centro de masa del
 *          sistema estrella-planeta
 * e        excentricidad de la órbita del planeta
 * m2sinI   masa del planeta por el seno del ángulo de inclinación de la órbita
 * m1       masa de la estrella
 * w        omega pequeña, argumento del periastro
 * noise    amplitud del ruido
 * tau      parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *          se usa en el caso de querer generar ruido rojo
 * arg      argumento switch para escoger el tipo de ruido a generar
 *****************************************************************************
 * Devuelve 0 si finaliza correctamente
 ****************************************************************************/
```

El tiempo inicial y final se introducen en días julianos (JD), el periodo en días, la velocidad del sistema en km/s, el valor de $m_2 \sin(i)$ en masas de Júpiter, la masa estelar en masas solares y el argumento del periastro en grados. Esta función genera una serie temporal y llama a la función de generación de ruido que describiré en el siguiente apartado. La variación de

la curva de velocidad radial con los parámetros orbitales se puede ver en el Apéndice: Fig. B.1a, B.1b y B.1c para la excentricidad, Fig. B.2a, B.2b y B.2c para el periodo, Fig. B.3a, B.3b y B.3c para la semiamplitud, Fig. B.4a, B.4b y B.4c para la velocidad radial del centro de masas del sistema, Fig. B.5a, B.5b y B.5c para el argumento del periastro.

#### 2.1.1.2. Generación del ruido

La generación del ruido la realiza la función `Noise()`. Esta contempla tres tipos de ruidos: un ruido no gaussiano, un ruido blanco gaussiano o un ruido rojo. En el caso del ruido no gaussiano como primera aproximación, se utilizan valores de una distribución pseudoaleatoria uniforme y se multiplica por un factor que controla la amplitud del ruido `noise`.

```
eta[i] = sqrt(noise) * (RandomUniform() - 0.5) * 2;
```

El ruido blanco gaussiano como veremos más adelante tiene una densidad espectral de potencia constante y que no depende de la frecuencia (espectro plano). Las señales son incorreladas y estadísticamente independientes. Este es el tipo de ruido que normalmente se contempla en la astroestadística. Para la generación de variables con distribución normal se utiliza el algoritmo #712 de la colección de algoritmos de Transactions on Mathematical Software de Association for Computing Machinery, posteriormente traducido a C por Paul Bourke [2]. Esta función `RandomGaussian()` devuelve número pseudoaleatorios distribuidos normalmente con una media y desviación estándar dadas.

```
wn[i] = sqrt(noise) * RandomGaussian (0.0, 1.0);
```

El ruido rojo, común en las ciencias atmosféricas y del clima, se puede modelar como un proceso gaussiano estacionario autoregresivo de primer orden, AR(1). En búsquedas de exoplanetas por velocidad radial se han detectado componentes de ruido correladas, o ruido rojo. Por ejemplo, Baluev (2012), según su análisis, encontró ruido rojo para GJ 581 y señaló que la aproximación comúnmente utilizada en astroestadística de tomar las señales con ruido incorrelado (ruido blanco gaussiano) puede ser pobre. El tratamiento de señales de velocidad radial con ruido rojo carece de un solución rutinaria práctica que funcione.

[2] University of Western Australia. `http://paulbourke.net/`.

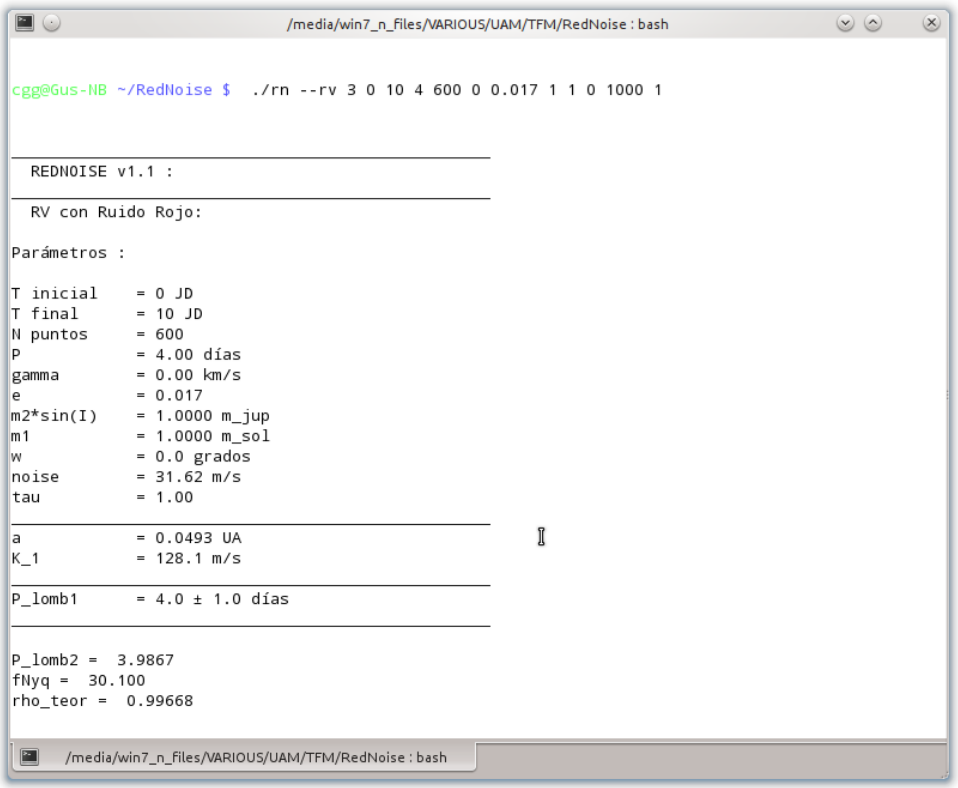



Figura 2.4: Captura de pantalla de `RedNoise` en modo consola para la generación de una curva de velocidad radial con ruido rojo.

En las Figs. B.6, B.7 y B.8 se muestran un ruido no gaussiano (distribución uniforme), un ruido blanco gaussiano (distribución normal) y un ruido rojo, respectivamente, todos ellos generados con `RedNoise` (600 puntos para apreciarlos mejor). Se puede notar como el ruido rojo puede llegar a "imitar" periodicidades, pues su distribución parece una mezcla ruidosa de señales periódicas ilusorias.

Un proceso AR(1) discreto $r$ para tiempos $t_i$ ($i$ = 1,2,...,N) con espaciado arbitrario se define así:

$$r(t_i) = \rho_i r(t_{i-1}) + \varepsilon(t_i), \tag{2.4}$$

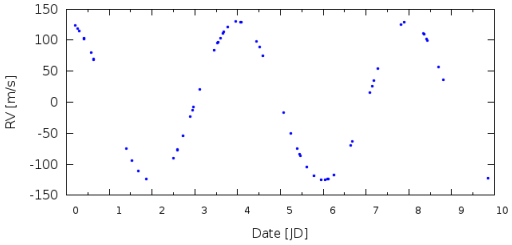
$$\rho_i = \exp(-(t_i - t_{i-1})/\tau) \tag{2.5}$$

donde $\tau$ es la escala de tiempo característica del proceso AR(1) o una medida de su "memoria" y $\varepsilon$ es un ruido blanco gaussiano con media 0 y varianza $\sigma_\varepsilon^2 \equiv 1-\exp(-2(t_i-t_{i-1})/\tau)$, lo que garantiza que el proceso sea estacionario y tenga varianza igual a 1 (Schulz & Mudelsee 2002). Después escalamos a la amplitud de ruido que queremos, de nuevo a través del parámetro `noise`. A continuación presento un extracto del código para la generación del proceso AR(1).

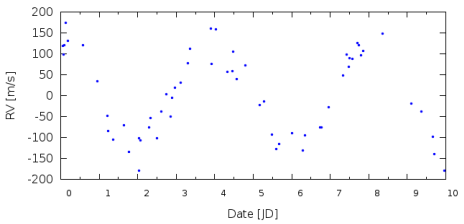
```
// Generación del ruido rojo, iteración 1
rn[0] = RandomGaussian(0.0,1.0)*sqrt(1-exp(-2*(t[1]-t[0])/tau));
wn[0] = rn[0];
rn_final[0] = sqrt(noise)*rn[0];
// iteración 2 en adelante
for (k = 1; k < N; k++)
{
    // Ruido blanco con varianza 1-exp(-2*(t[k]-t[k-1])/Tau
    wn[k] = RandomGaussian(0.0,1.0)*sqrt(1-exp(-2*(t[k]-t[k-1])/tau));
    Ro[k] = exp(-(t[k]-t[k-1])/tau);
    rn[k] = (Ro[k]*rn[k-1])+wn[k];
    rn_final[k] = sqrt(noise)*rn[k];
}
```

En la Fig 2.4 se muestra una captura de pantalla de la salida en modo consola de `RedNoise`. En la Fig. 2.5 se pueden ver curvas de velocidad radial para un Júpiter caliente con diferentes ruidos. A continuación muestro un resumen de la función `Noise()`:

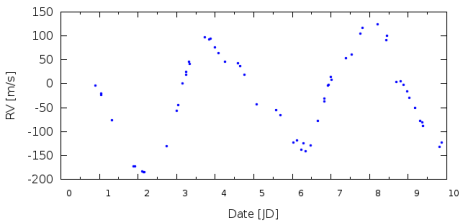
```
/*****************************************************************************
 * FUNCIÓN: Noise
 *****************************************************************************
 * Función para agregar ruido a la sinusoide o a la RV modelada. El ruido puede
 * ser de 3 tipos: ruido no gaussiano, ruido blanco gaussiano y ruido rojo
 *****************************************************************************
 * Argumentos:
 * arg      argumento switch para controlar el flujo de la función y generar
 *          alguno de los tres tipos de ruidos y sumarlos a la señal
 * N        número de puntos de la serie temporal de la curva
 * noise    amplitud del ruido
 * tau      parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *          se usa en el caso de querer generar ruido rojo
 * *t       array con los tiempos de la serie temporal
 * *rv      array con las medidas de RV de la curva o con la sinusoide
 *****************************************************************************
 * Devuelve 0 en caso de finalizar correctamente
 *****************************************************************************/
```



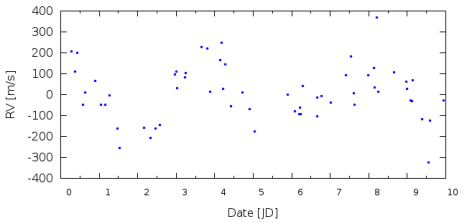

(a) Sin ruido.



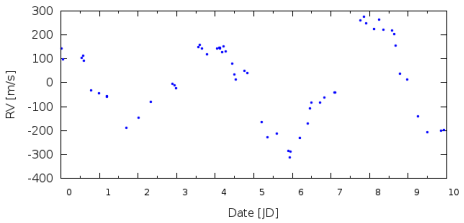

(b) Con ruido blanco gaussiano con amplitud de 30 m/s.




(c) Con ruido rojo con amplitud de 30 m/s.




(d) Con ruido blanco gaussiano con amplitud de 100 m/s.




(e) Con ruido rojo con amplitud de 100 m/s.

Figura 2.5: Curvas de velocidad radial para un Júpiter caliente. Versión ampliada en el Apéndice en Figs. B.9 y B.10.

### 2.1.2. Periodograma

En Astronomía son comunes las series temporales con vacíos debidos a los ciclos del día y la noche, a los ciclos orbitales de los satélites y/o a la concesión de tiempo de observación en los telescopios. El análisis espectral revela la varianza de una señal a diferentes frecuencias. Matemáticamente, el análisis espectral está relacionado con una herramienta llamada transformada o análisis de Fourier. Esta depende de varias suposiciones raramente cumplidas en datos astronómicos: datos equiespaciados en tiempo de duración infinita con una alta frecuencia de muestreo (frecuencia de Nyquist), ruido gaussiano, periodicidad de una sola frecuencia con forma sinusoidal y comportamiento estacionario. Estas condiciones se violan en diferentes problemas astronómicos. El periodograma [3] de Lomb-Scargle es un equivalente al ajuste por mínimos cuadrados de ondas sinusoidales, que apunta a la detección eficiente y confiable de una señal periódica en el caso de observaciones no equiespaciadas y en presencia de ruido (Scargle 1982; Horne & Baliunas 1986; Cumming 2004; Baluev 2008). En `RedNoise` se realizó una implementación en C del Lomb-Scargle en la función `Powerespectrum()`:

```
/*****************************************************************************
 * FUNCIÓN: Powerspectrum
 *****************************************************************************
 * Función con el LOMB_SCARGLE para calcular el espectro de potencias de una RV,
 * o en general de una serie temporal no equiespaciada en tiempo
 *****************************************************************************
 * Argumentos:
 * file archivo que contiene la serie temporal con la RV
 * arg   argumento switch para el flujo de la función, permite guardar en dos
 *       archivos diferentes dependiendo si es el espectro de una RV o un ruido
 *****************************************************************************
 * Devuelve el periodo máximo en el espectro. También escribe en archivo la
 * serie temporal del espectro
 *****************************************************************************/
```

También se incluyó una versión del Lomb-Scargle en MATLAB/Octave directamente basada en el algoritmo descrito en el "Numerical Recipes".

```
/*****************************************************************************
 * FUNCIÓN: Powerspectrum_lomb
 *****************************************************************************
 * Función que llama a la función de MATLAB LOMB basada en numerical recipes
 *****************************************************************************
 * Argumentos:
 * arg     argumento switch para el flujo de la función, permite imprimir
 *         archivos diferentes si es una serie RV o del ruido extraido
 *****************************************************************************
 * Devuelve 0 si finaliza correctamente
 *****************************************************************************/
```

[3] Espectro de potencias o simplemente espectro.

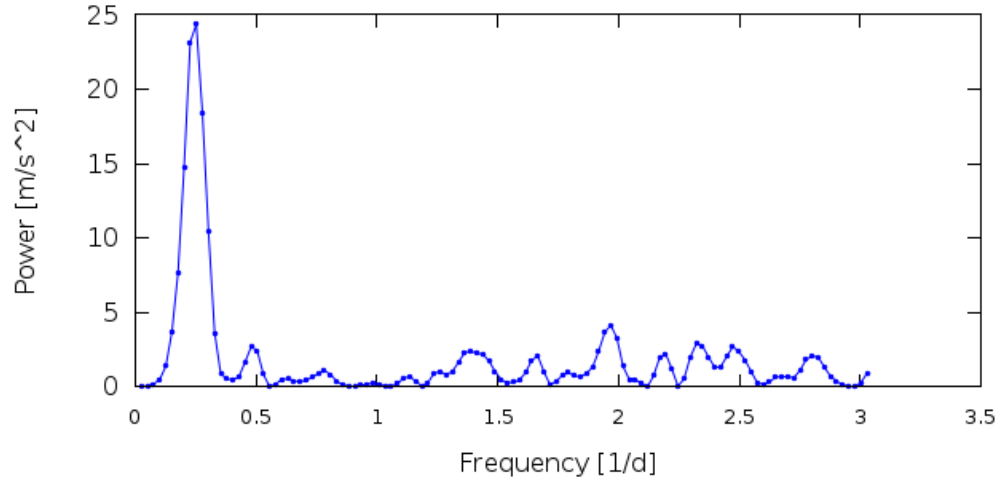



Figura 2.6: Periodograma de Lomb-Scargle para una curva de velocidad radial de un exoplaneta tipo Júpiter caliente, sin ruido.

Un ejemplo de periodograma para el caso de las curvas de velocidad radial del Júpiter caliente vistas anteriormente se tiene en la Fig. 2.6

### 2.1.3. Extracción del ruido rojo y ajuste de sus parámetros

#### 2.1.3.1. Ajuste de los parámetros orbitales

Después de implementada la curva de velocidad radial con el ruido deseado nos concentramos en el ruido rojo. Para extraer el ruido rojo de nuestra señal empezamos por hacer el ajuste de los parámetros orbitales o, en otras palabras, extraer los parámetros orbitales del exoplaneta de los datos de velocidad radial (que no es más que el problema inverso a lo hecho en la sección anterior). Dado que las ecuaciones de la velocidad radial son no lineales, se han implementado esquemas de búsqueda de mínimos $\chi^2$ locales, como el método de Lavenberg-Marquardt, que es ampliamente usado para el ajuste de órbitas astrométricas y de velocidad radial. Implementaciones de este método se encuentran en la literatura por Press el al. (2007, mrqmin) y por Wright & Howard (2009, RVLIN) en IDL. Por el lado del enfoque Bayesiano está el método de Monte Carlo (Markov Chain Monte Carlo, MCMC). Una implementación para el ajuste de sistemas de uno o dos planetas es EXOFIT por Balan & Lahav (2009). Un código aún más reciente se debe a Eastman et

al. (2013, EXOFAST). También se han intentado utilizar algoritmos genéticos pero han sido ineficientes para encontrar la mejor solución del ajuste. En este trabajo se utilizó EXOFIT para el ajuste de los parámetros orbitales.

#### 2.1.3.2. Extracción del ruido rojo y ajuste del espectro del ruido

Después de obtenidos los parámetros orbitales procedemos a generar una nueva curva a partir de estos, pero esta vez sin ruido, que corresponderá a la señal de la velocidad radial debida al planeta. Luego recuperamos la señal del ruido rojo que, aunque viene mezclada con un poco de señal periódica, nos sirve para el ajuste de los parámetros de su espectro. La función de `RedNoise` que se encarga de la generar esta curva sin ruido y sustraerla a la señal inicial (con ruido rojo) es `RnExtract()`. Después de restar las señales y quedar con la señal del ruido rojo tomamos su espectro con el Lomb-Scargle. Se implementó una función Central Moving Average para el suavizado del espectro (que está incluida en `RedNoise`) que finalmente no se utilizó para evitar complicaciones al calcular el test de significación.

El diagrama de salida que genera `RedNoise` con su función `RnExtract()` se puede apreciar en la Fig. 2.7. En el panel superior está la curva de velocidad radial con ruido rojo inicialmente modelada, en el panel central su diagrama en fase, y superpuesta la curva de velocidad radial sin ruido que acabamos de generar (a partir del ajuste de los parámetros orbitales). Más abajo, la sección $O - C$ (Observada – Calculada) es la resta de estas dos señales, que corresponde al ruido rojo extraído. En el panel inferior están el periodograma de la señal con ruido en azul, y en rojo el periodograma del ruido rojo aislado.

```
/*****************************************************************************
 * FUNCIÓN: RnExtract
 *****************************************************************************
 * Función que extrae el ruido rojo de la curva RV, utilizando los parámetros
 * del ajuste que arroja EXOFIT
 *****************************************************************************
 * Argumentos:
 * P        periodo (T1 en la nomenclatura de EXOFIT)
 * K_1      semiamplitud de la RV (K1 en la nomenclatura de EXOFIT)
 * gamma    RV_0, velocidad sistémica o velocidad radial del centro de masa del
 *          sistema estrella-planeta (V en la nomenclatura de EXOFIT)
 * e        excentricidad de la órbita del planeta (e1 en EXOFIT)
 * w        omega pequeña o argumento del periastro (w1 en EXOFIT)
 * ventana  ventana de suavizado para el algoritmo CMA
 *****************************************************************************
```

Como ya tenemos la señal del ruido rojo, procedemos al ajuste de su espectro. Esto lo hace `RedNoise` a través de su función `RnFit()`, que a su vez llama a una función en MATLAB que realiza el ajuste a la siguiente función:

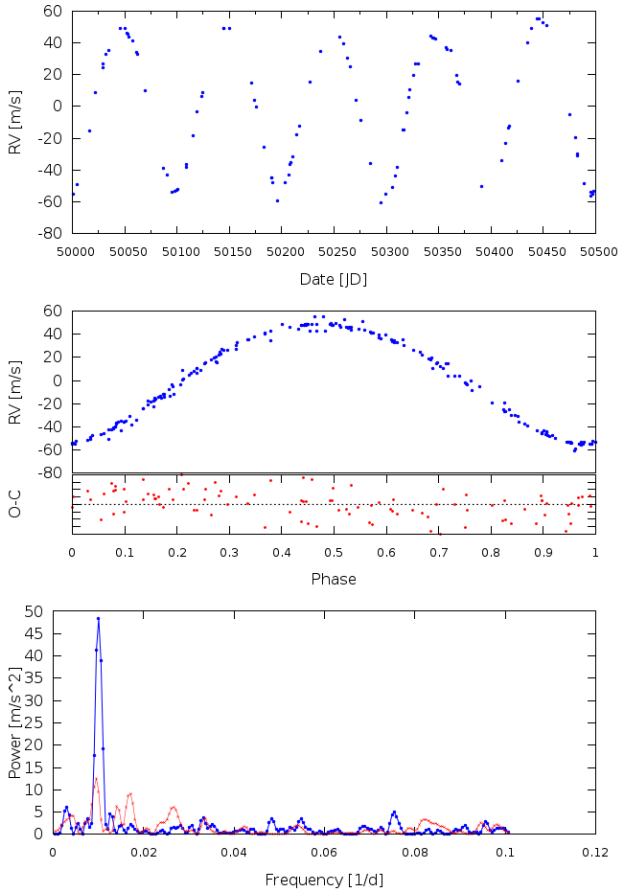



Figura 2.7: Salida gráfica de la función `RnExtract()`.

$$S(f) = \frac{A}{1 + \rho^2 - 2\rho\cos(\pi f / F_{Nyq})}, \tag{2.6}$$

que es la función de densidad espectral teórica del proceso AR(1) (Wilks 2006). El parámetro que nos interesa es $\rho$ que está relacionado con $\tau$ o la memoria del ruido rojo. Cabe anotar que para $\rho$ igual a cero obtenemos ruido blanco. $F_{Nyq}$ es la frecuencia de sampleado o frecuencia de Nyquist. Para datos equiespaciados la frecuencia de Nyquist se toma como $0{,}5\delta t$, donde $\delta t$ es el paso temporal de sampleado. En nuestro caso, con datos no equiespaciados, la tomamos como $0{,}5\delta t_{ave}$, donde $\delta t_{ave}$ es la media de los pasos temporales. En la Fig. 2.8 se puede ver un ejemplo de tal ajuste del espectro del ruido rojo, que es justo comos esperaba, una función exponencial que decae dependiendo del parámetro $\tau$.

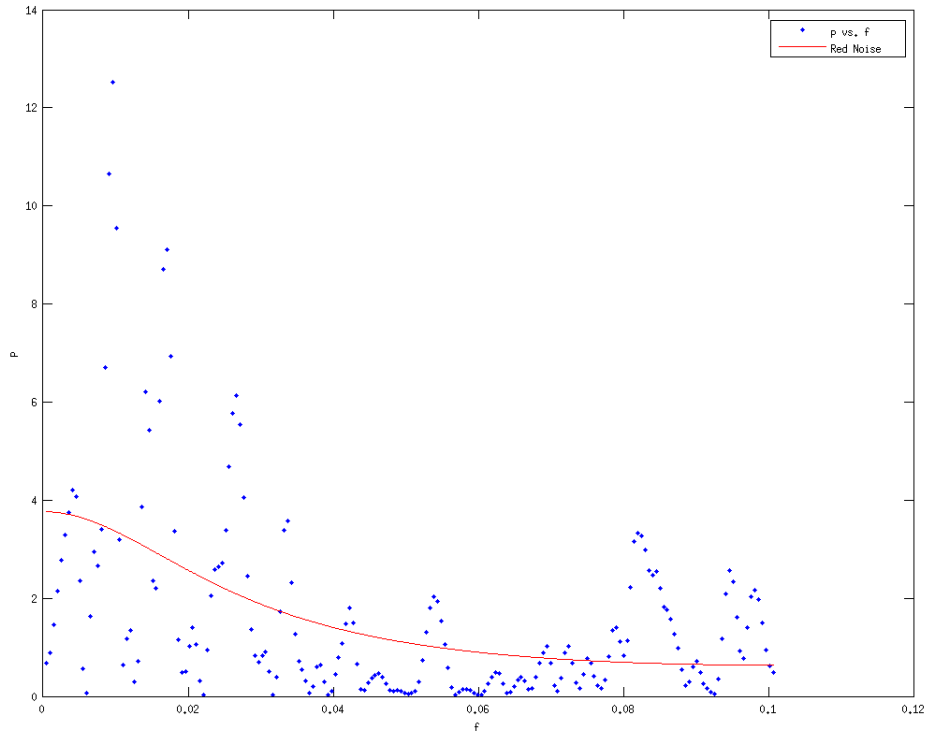



Figura 2.8: Ajuste del espectro del ruido rojo extraído.

### 2.1.4. Test de significación

Consideramos que la hipótesis nula de que la amplitud al cuadrado $C_k^2$ a la frecuencia $f_k$ es significativamente mayor que la amplitud del espectro (de ruido rojo) a esa misma frecuencia $S_0(f_k)$ será rechazada al nivel $\alpha$ si:

$$C_k^2 \geq \frac{S_0(f_k)}{\nu}\chi_\nu^2(1-\alpha), \tag{2.7}$$

donde $\chi_\nu^2(1-\alpha)$ denota los cuantiles del ala derecha de la distribución $\chi^2$, $\nu$ son los grados de libertad de la $\chi^2$ (en nuestro caso $\nu$ es igual a dos; sería mayor si utilizáramos algún tipo de suavizado espectral como el Central Moving Average). Como queremos realizar un test múltiple sobre todas las frecuencias independientes, debemos modificar la ecuación anterior con $\alpha' = \alpha/M$ (donde M es el número de frecuencias independientes, que es aproximadamente el número de frecuencias del espectro hasta la frecuencia de Nyquist), resultando un test más riguroso. La función de `RedNoise` que se encarga de esto es `RnTestMultiple()`:

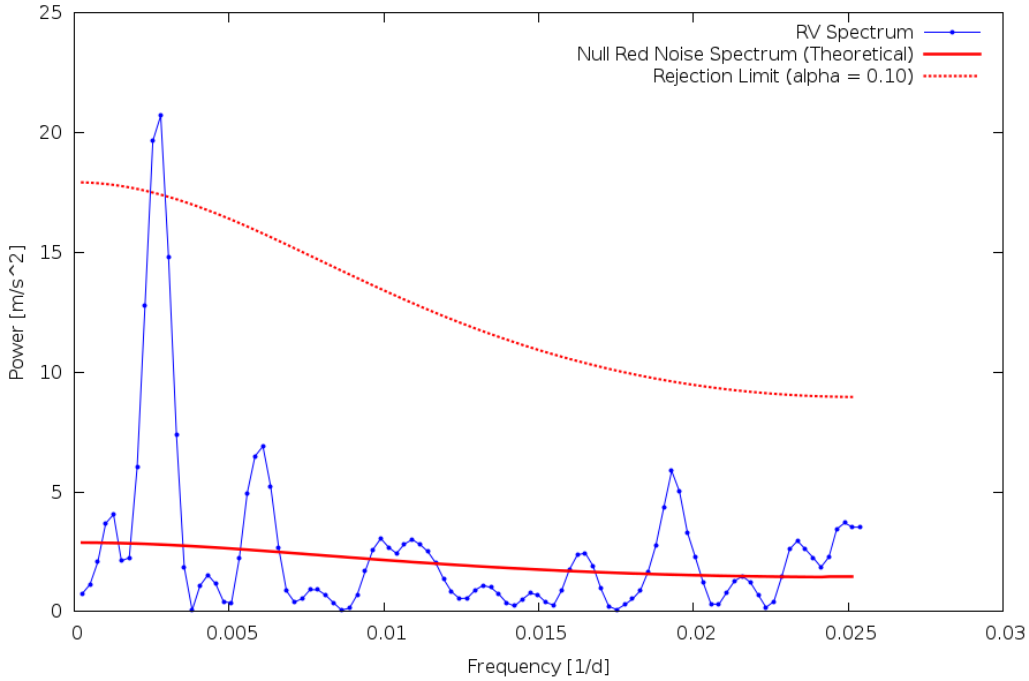



Figura 2.9: Diagrama con el test de significación que genera `RedNoise`.

```
/*****************************************************************************
 * FUNCIÓN: RnTestMultiple
 *****************************************************************************
 * Función para calcular el test de significación de los picos en el espectro
 * de la RV. Esta es la versión para un testo múltiple
 *****************************************************************************
 * Argumentos:
 * Rho      parámetro del ruido rojo. está relacionado con su memoria, depende
 *          de tau
 * A        parámetro del ruido rojo. factor relacionado con su amplitud,
 *          depende de noise
 * fNyq     frecuencia de Nyquist (sampleado) en la señal de la RV modelada
 * alpha    incertidumbre para el test de significación
 *****************************************************************************
 * Devuelve una estructura que contiene 3 arrays, uno para la frecuencia, otro
 * para la potencia del espectro teórico y otro para el límite de significación
 * También escribe estos arrays al archivo "data_power_noisig_test.dat"
 *****************************************************************************/
```

`RnTestMultiple()` genera un diagrama con el espectro de la señal modelada, la curva de velocidad radial con ruido rojo, el espectro teórico del ruido rojo extraído y el límite de significación a un determinado nivel. Un ejemplo se puede ver en la Fig. 2.9.

# Capítulo 3

# Resultados y discusión

## 3.1. Modelado de exoplanetas de diferentes tipos

Como mostramos en la metodología, se consiguió realizar un test de significación contra la hipótesis de un ruido rojo en la señal de velocidad radial. Partiendo de este resultado, procederemos a modelar curvas de velocidad radial para planetas de diferentes tipos, júpiteres y neptunos calientes, supertierras y exotierras. Para este último caso nos detendremos a analizar donde está el umbral de detección (en una señal con ruido rojo) utilizando el test de significación antes propuesto.

Utilizamos como modelo de supertierra a GJ 876d. Los parámetros que nos interesan de este sistema han sido tomados de `exoplanets.org` y se encuentran en la Tabla. 3.1.

El $\tau$ o memoria del ruido rojo se escoge de tal forma que sea del orden de magnitud del $\Delta T_{\text{medio}}$, lo que garantiza que sea suficientemente rojo. Cuando $\tau$ tiende a cero, el ruido rojo tiende a blanco gaussiano.

| Planeta | $P$ [d] | $e$ | $M_2 \sin(i)$ $[M_{\text{Jup}}]$ | $M_1$ $[M_\odot]$ | $\omega$ [deg] |
|---|---|---|---|---|---|
| GJ 876 d | 1.93778 | 0.207 | 0.0184 | 0.3 | 234 |

Tabla 3.1: Tabla con los valores orbitales de los sistemas planetarios a modelar.

### 3.1.1. Caso de una supertierra

Para el caso de una supertierra, se usaron de los parámetros de GJ 876d, quedando como está a continuación:

```
--------------------------------------------------
T inicial    = 0 JD
T final      = 10 JD
N puntos     = 50
P            = 1.94 días
gamma        = 0.00 km/s
e            = 0.207
m2*sin(I)    = 0.0184 m_jup
m1           = 0.3000 m_sol
w            = 234.0 grados
noise        = 1.000 m/s
tau          = 0.20
--------------------------------------------------
a            = 0.0204 UA
K_1          = 6.8 m/s
--------------------------------------------------
P_lomb1      = 2.0
P_lomb2      = 1.8927
--------------------------------------------------
```

Seguimos los pasos de la metodología que serían:

- hacer el ajuste de los parámetros orbitales de la curva de velocidad radial, usando el EXOFIT,
- generar una nueva curva de velocidad radial sin ruido a partir de estos parámetros,
- extraer el ruido, restando estas señales,
- tomar el espectro de este ruido, y ajustarlo a la función de densidad espectral teórica del proceso AR(1),
- y, con los parámetros del ajuste generar el test de significación y evaluar nuestro pico.

Después de aplicados los pasos encontramos que hay detección clara de la supertierra ante la presencia de un ruido rojo de 1 m/s con un 99 % de confianza (ver Figs. 3.1, 3.2 y 3.3).

### 3.1.2. Caso de una exotierra

Para el modelado de una exotierra se han usado los parámetros terrestres. El primer caso modelado fue el de una curva de velocidad radial con ruido rojo de una exotierra con los siguientes parámetros:

```
-------------------------------------------------
T inicial    = 0 JD
T final      = 1000 JD
N puntos     = 50
P            = 365.24 días
gamma        = 0.00 km/s
e            = 0.017
m2*sin(I)    = 0.0032 m_jup
m1           = 1.0000 m_sol
w            = 0.0 grados
noise        = 0.031 m/s
tau          = 20.00
-------------------------------------------------
a            = 1.0000 UA
K_1          = 0.1 m/s
-------------------------------------------------
P_lomb1      = 370.3
P_lomb2      = 358.00
-------------------------------------------------
```

El efecto que ejerce esta exotierra en su sol es de 10 cm/s. La amplitud del ruido es de 3,1 cm/s, casi un tercio de la semiamplitud de la curva. El Lomb-Scargle encuentra el período principal con pequeño error. A continuación se aplica la metodología de extracción (ver Fig. 3.4 y ajuste del espectro del ruido rojo (ver Fig. 3.5) para generar el test de significación. El resultado obtenido es que sólo se tiene una detección alrededor de los $2\sigma$, o 95 %, $\alpha = 0.05$). Con el 99 % de certeza no hay detección (ver Figs. 3.6 y 3.7).

Es interesante mirar como resulta una realización con menos ruido, ahora de 1 cm/s y con 80 puntos (lo que equivaldría a más observaciones de velocidad radial de ser un caso real). Esto último se hace para elevar la significación estadística del pico del armónico principal (ver Fig. 3.8). Después de aplicar nuevamente la metodología encontramos que el pico se detecta a 99 % de confianza sin problema, pero notamos que en esta realización el ajuste de $\rho$ resultó menos preciso, causando que el espectro teórico del ruido rojo y

posteriormente el test de significación sean más planos. Esto favorece a los picos que se encuentran a bajas frecuencias (ver Figs. 3.9 3.10).

Generamos a continuación una realización idéntica (con exactamente los mismos parámetros), buscando que esta vez resulte mejor el ajuste del espectro del ruido rojo (ver Figs. 3.11 y 3.12). Todo funciona según lo esperado y se tiene detección clara a 99 % (ver Figs. 3.13 y 3.14).

Las realizaciones anteriores para una curva de velocidad radial con ruido rojo, de una estrella tipo solar con un planeta tipo tierra a su alrededor nos permiten deducir que el ruido máximo para una detección al 99 % de confianza del pico principal (la exotierra) del periodograma de Lomb-Scargle está alrededor de 1 cm/s. Teniendo en cuenta que la semiamplitud de tal curva es de 10 cm/s, vemos que con un ruido de una décima parte de la semiamplitud detectamos el pico de la exotierra a un 99 % de confianza o $3\sigma$. Estas pruebas nos permiten ver que la metodología es sensible al ajuste del parámetro $\rho$ del espectro del ruido rojo, que hace que la caída de este sea menos pronunciada de lo esperado para $\rho$ menores al $\rho$ teórico y que el test sea menos estricto a bajas frecuencias. La relación entre $\rho$ y $\tau$ es la siguiente: `Rho_ruido = exp (-deltaTave/Tau)`. De aquí vemos que un ruido con más memoria o más rojo (con un $\tau$ mayor que conlleva a un $\rho$ mayor) afecta a qué tan pronunciada sea la caída del espectro, lo que debería afectar sobre todo a picos en bajas frecuencias, o planetas con períodos grandes.

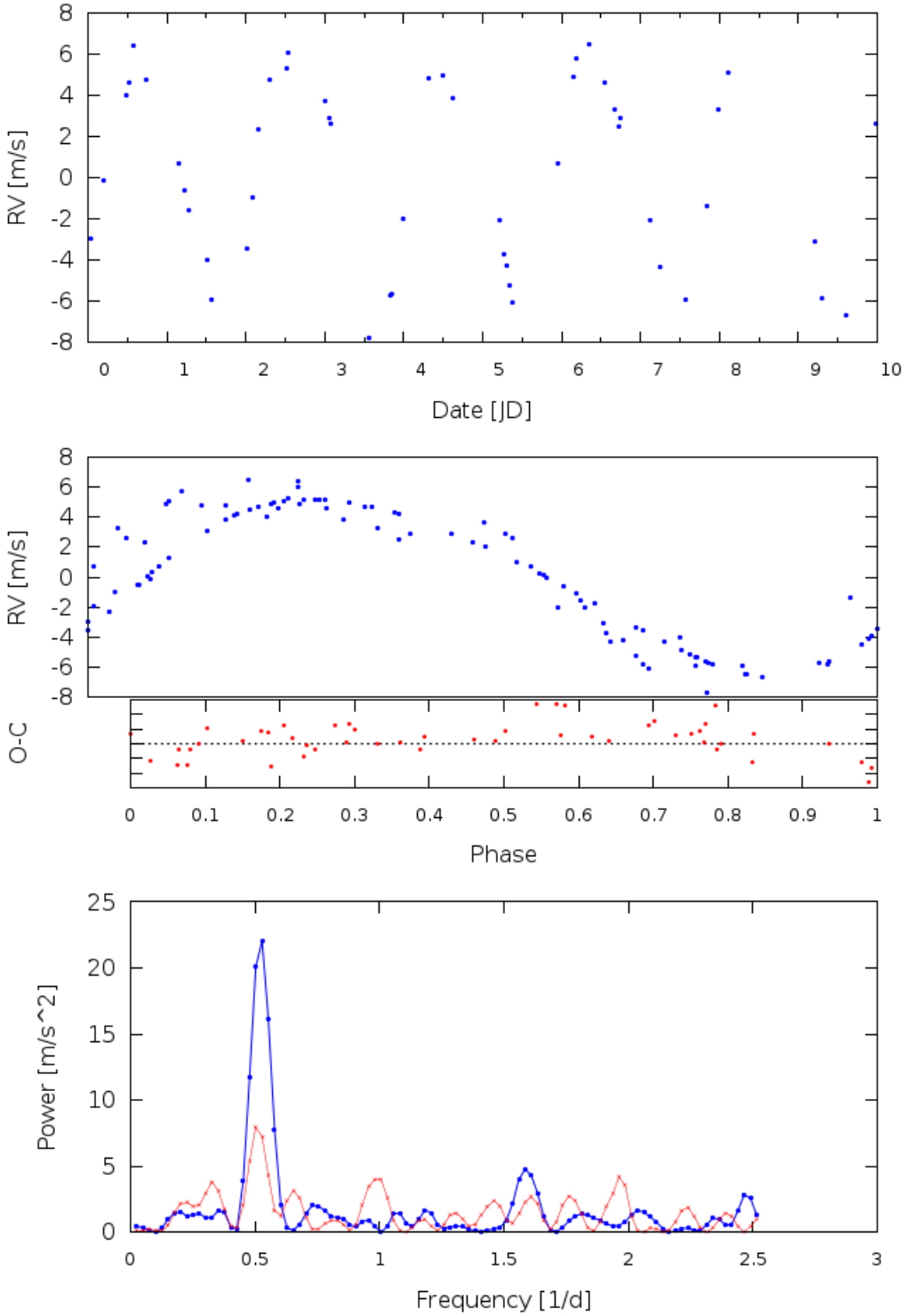



Figura 3.1: Curva de velocidad radial modelada de una supertierra (panel superior), diagrama en fase de la curva y ruido (panel central) y periodogramas de la señal y el ruido (panel inferior).

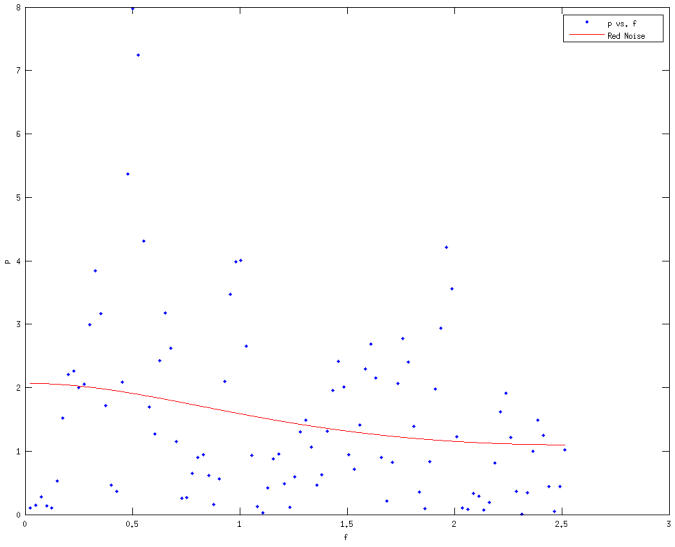



Figura 3.2: Ajuste del espectro del ruido rojo.

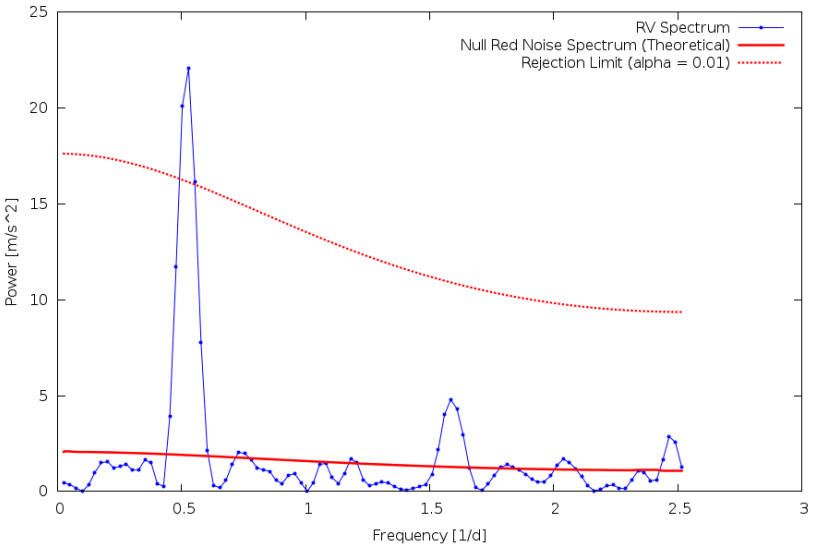



Figura 3.3: Test de significación para $\alpha = 0.01$.

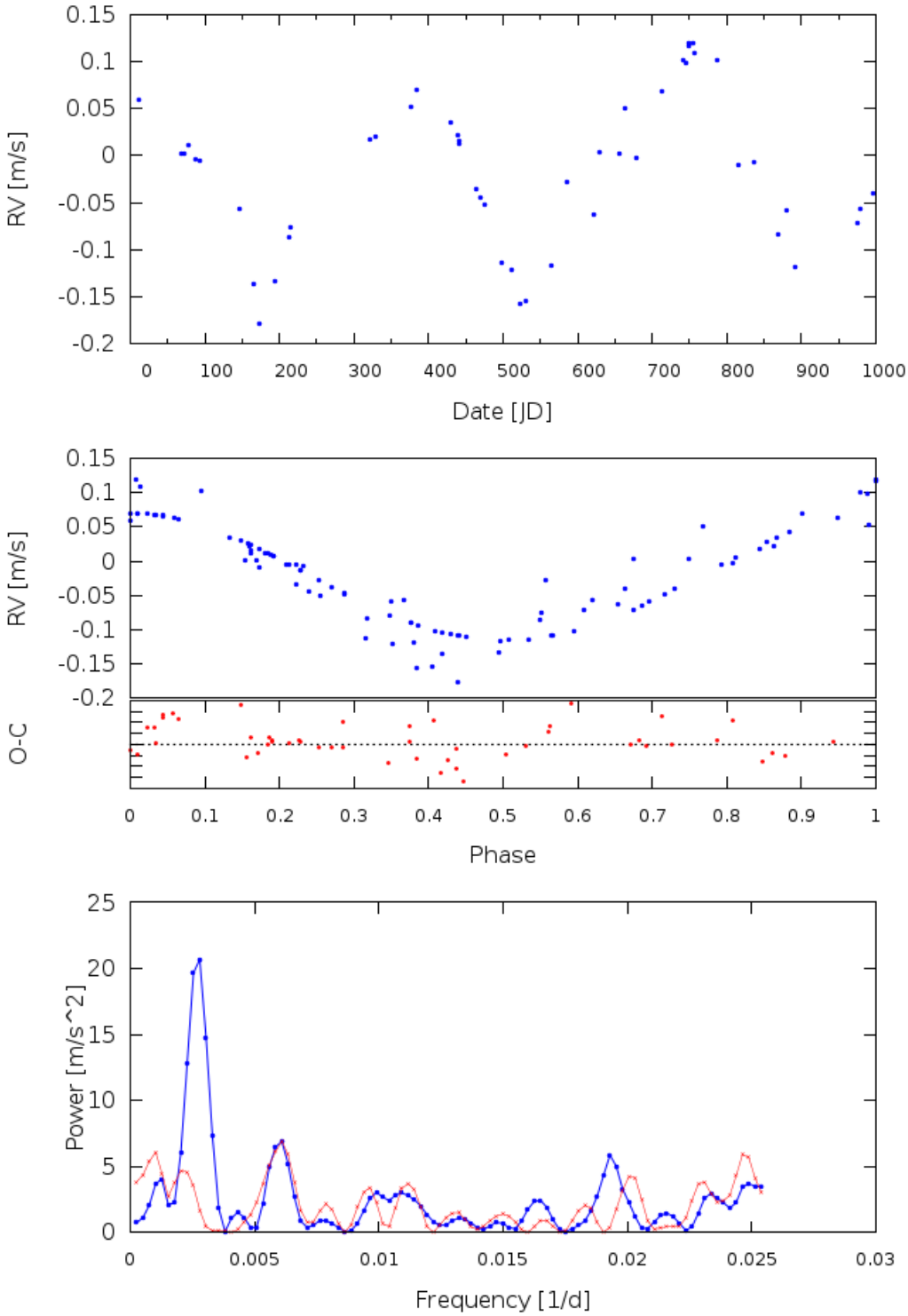



Figura 3.4: Extracción del ruido rojo de la señal de velocidad radial de una exotierra con ruido de 3.1 cm/s.

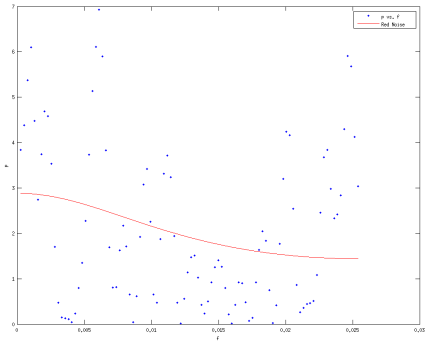


Figura 3.5: Ajuste del espectro del ruido.

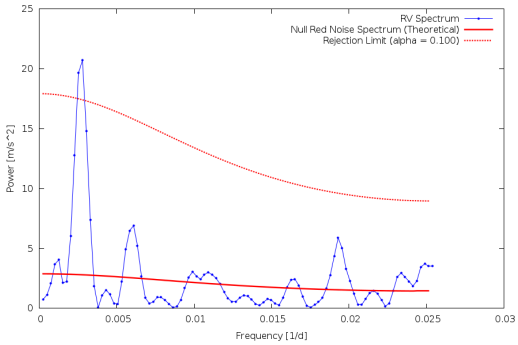



Figura 3.6: Test de significación para $\alpha = 0.05$.

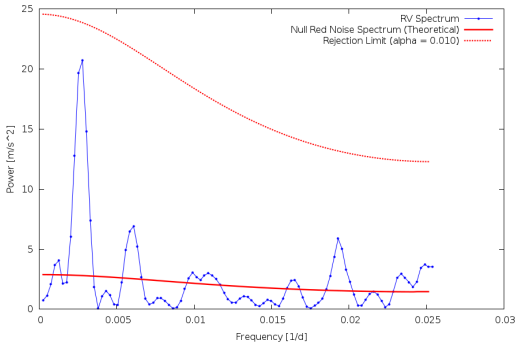



Figura 3.7: Test de significación para $\alpha = 0.01$.

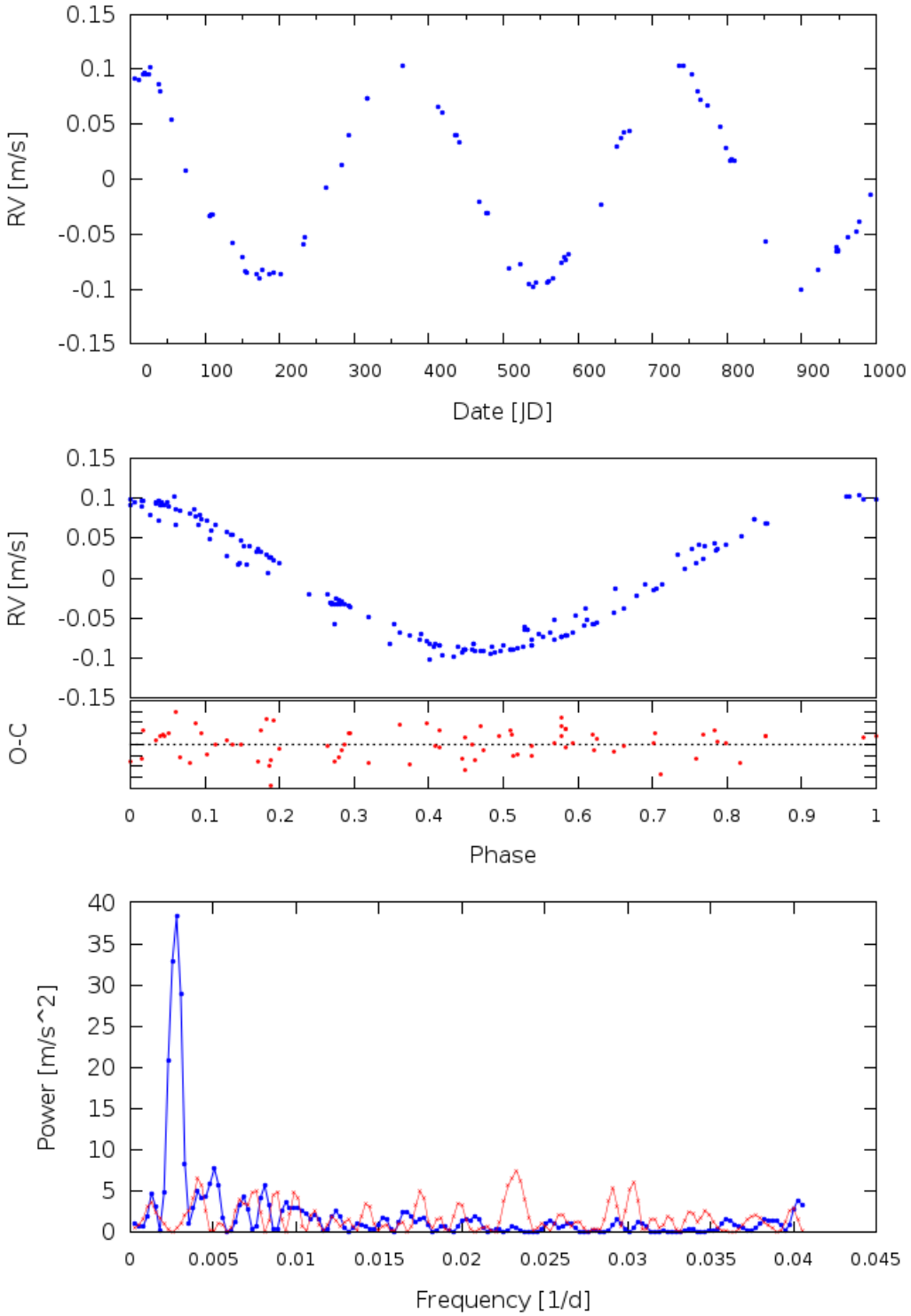



Figura 3.8: Extracción del ruido rojo de la señal de la velocidad radial de una exotierra con ruido de 1 cm/s.

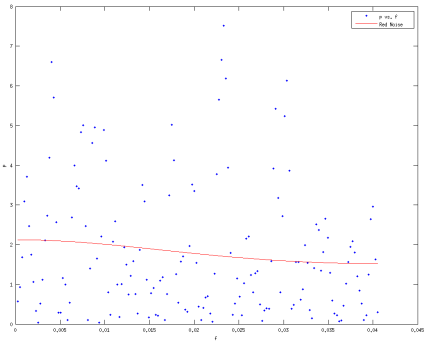


Figura 3.9: Ajuste del espectro del ruido. Se nota que no es óptimo, y resulta un poco más plano de la esperado.

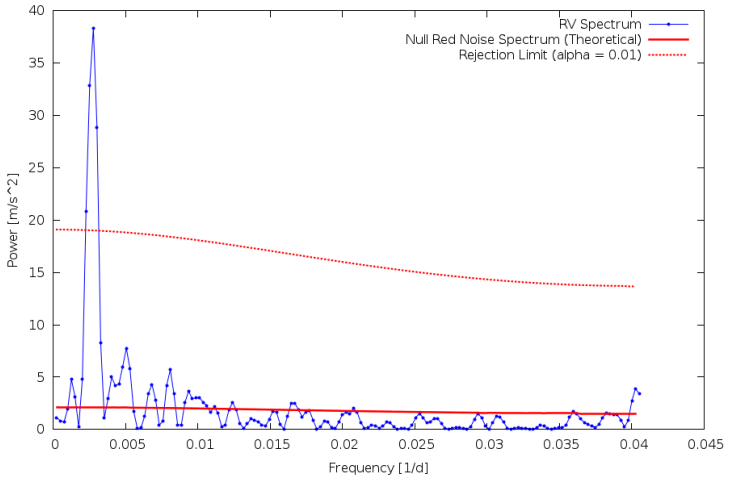



Figura 3.10: Test de significación para $\alpha = 0.01$.

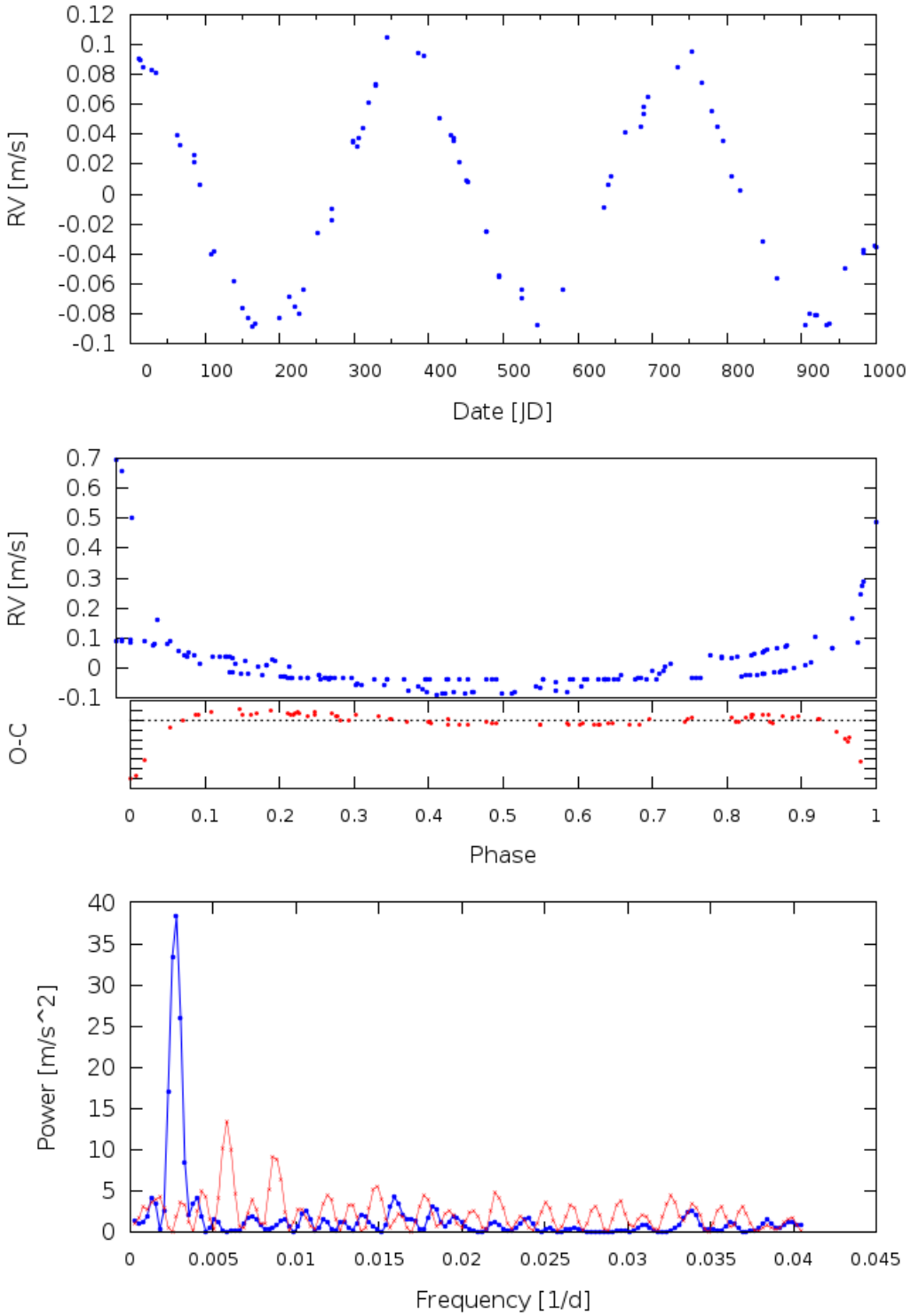



Figura 3.11: Extracción del ruido rojo de la señal para la exotierra con el mismo ruido del caso anterior.

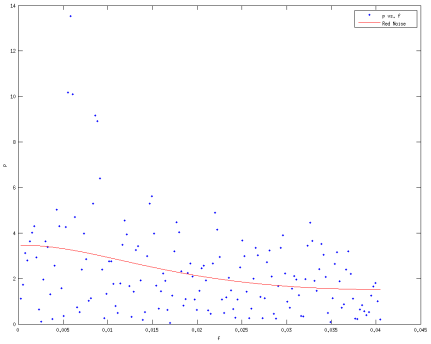


Figura 3.12: Ajuste del espectro del ruido de la señal de la velocidad radial de una exotierra con ruido de 1 cm/s. Se consigue un mejor ajuste de $\rho$, ahora la curva exponencial está más pronunciada.

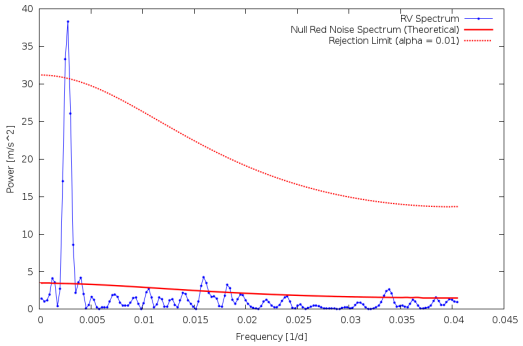



Figura 3.13: Test de significación para $\alpha = 0.01$.

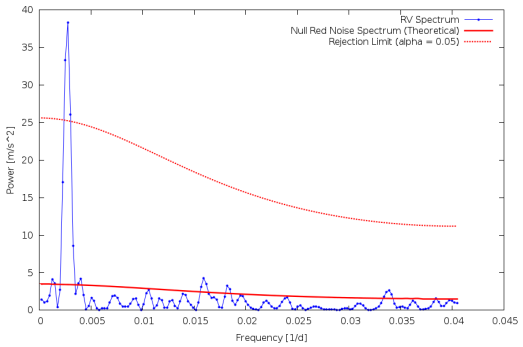



Figura 3.14: Test de significación para $\alpha = 0.05$.

# Capítulo 4

# Conclusiones y trabajo futuro

## 4.1. Conclusiones

Se ha buscado diseñar un test de significación para el caso de la presencia de ruido rojo en las señales de velocidad radial a partir de una metodología de modelado de este tipo de ruido. Con tal fin se ha desarrollado un programa que automatiza gran parte de este proceso, desde la generación de curvas de velocidad radial sintéticas con distintos tipos de ruido, hasta la generación de gráficos (con los test de significación) que permiten hacer una estimación sobre la significación de los picos en el periodograma de nuestra curva de velocidad radial.

Es importante resaltar que es la primera metodología (de la que tenemos conocimiento) que utiliza un modelo autoregresivo de primer orden para el modelado del ruido en una señal de velocidad radial estelar que presuntamente contiene un ruido rojo. Después de muchas realizaciones de curvas sintéticas para diversas configuraciones de sistemas exoplanetarios, se hacen conclusiones sobre la relación entre la amplitud del ruido rojo en una señal con la confianza en la detección del un pico en el periodograma de dicha señal. En este punto nos centramos en las exotierras; para uno de estos planetas encontramos que el ruido máximo para una detección al 99 % de confianza del pico principal en el espectro de Lomb-Scargle de la señal de velocidad radial (la exotierra) está alrededor de 1 cm/s. Teniendo en cuenta que la semiamplitud de tal curva es de 10 cm/s, vemos que con un ruido de una décima parte de la semiamplitud logramos detectar el pico de la exotierra a un 99 % de confianza o $3\sigma$. Esta precisión de velocidad radial se podrá alcanzar con los nuevos telescopios de clase 30–40 m con sistemas de calibración de longitud de onda avanzados como los *Laser Combs* (el espectrógrafo ESPRESSO en

el Very Large Telescope espera tener una precisión de velocidad radial de 10 cm/s).

## 4.2. Trabajo futuro

Si bien los objetivos principales del trabajo, que eran el desarrollo de un programa para el modelado de curvas radiales con ruidos de distinto tipo, el desarrollo de una metodología para el modelado del ruido rojo, y su posterior extracción de la señal para crear un test de significación de los picos del periodograma de la curva de velocidad radial, se cumplieron, este trabajo tiene mucho potencial para ser continuado. Por el lado del código, se requieren muchas mejoras, incluso he contemplado la idea de reimplementar todos los algoritmos en un lenguaje orientado a objetos como Python, lo que traería ciertas ventajas, como la facilidad en mantener y expandir un código grande. También podría implementar un algoritmo para el ajuste de los parámetros orbitales y encontrar mejores versiones del algoritmo para generar el periodograma de Lomb-Scargle o tratar de corregir el "bias" para altas frecuencias que introduce la aplicación del Lomb-Scargle a datos no equiespaciados.

En cuanto al análisis del ruido en curvas de velocidad radial, el siguiente paso sería aplicar la metodología a curvas de velocidad radial reales. En este proceso se pueden incorporar técnicas de tratamiento de señales no equiespaciadas como el prelavado ("pre-whitening") para la eliminación de las componentes de Fourier dominantes. Así se podrían estudiar sistemas con múltiples planetas, como el controvertido GJ 581 sobre el que diferentes estudios arrojan un número de entre tres y cinco planetas (Vogt et al. 2010, 2012; Forveille et al. 2011; Baluev 2012).

También sería muy interesante estudiar otro método de ajuste de datos con ruido rojo, como el propuesto por Baluev (2013), quien recientemente (hace sólo un par de semanas) ha publicado un artículo titulado *"Planet-Pack: a radial-velocity time-series analysis tool facilitating exoplanets detection, characterization, and dynamical simulations"*. En este trabajo describe un programa de su autoría, que hace público, llamado PlanetPack donde implementa (sólo una de las funciones del programa) un ajuste considerando ruido correlado (rojo). Además dice incluir versiones mejoradas del periodograma de Lomb-Scargle y métodos para las tareas de determinación de significación estadística.

# Agradecimientos

*Ad astra per aspera*

Séneca, el joven

Quisiera agradecer a la Universidad Autónoma de Madrid por el financiamiento, a través de la Beca Campus de Excelencia UAM+CSIC, para cursar el Máster Interuniversitario en Astrofísica de las Universidades Complutense y Autónoma de Madrid. A mi amada familia, por estar siempre e incondicionalmente presente. Por último, también quisiera agradecer al Dr. Jose Caballero y al Dr. Luis Dinis por permitirme desarrollar este trabajo de fin de máster e iniciación a la investigación bajo su supervisión, y además por el continuo apoyo en el proceso de su realización.

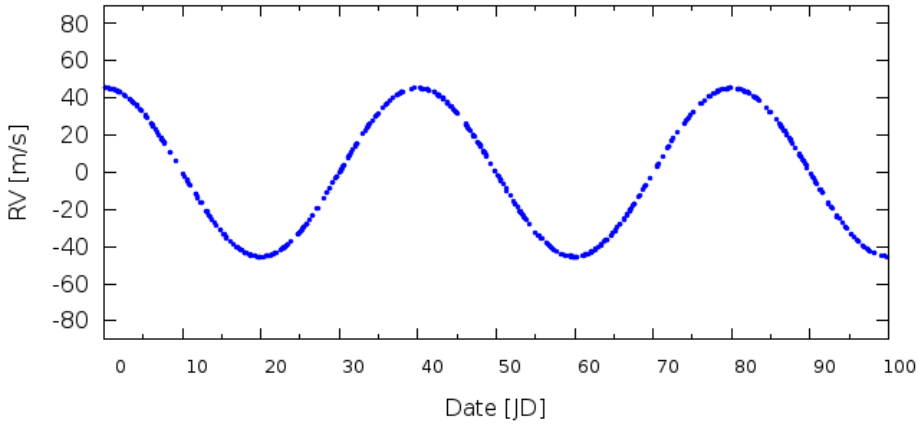
# Apéndice A

# Bibliografía

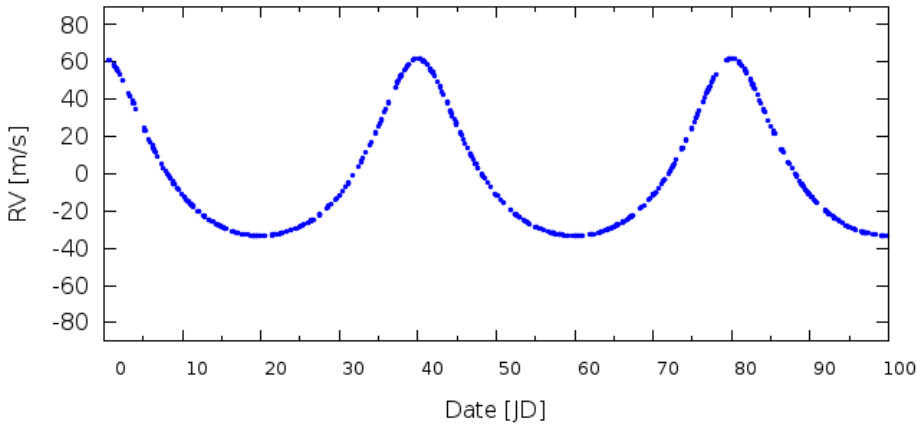

Balan, S. T. & Lahav, O. 2009, MNRAS, 394, 1936
Baluev, R. V. 2008, MNRAS, 385, 1279
Baluev, R. V. 2012, MNRAS, 422, 2372
Baluev, R. V. 2013, MNRAS, 431, 1600
Charbonneau, D. et al. 2009, Nature, 462, 891
Cumming, A. 2004, MNRAS, 354, 1165
Eastman, J., Gaudi, B. S. & Agol, E. 2013, PASP, 125, 8
Forveille, T. et al. 2011, A&A, submitted (eprint arXiv:1109:2505)
Haghighipour, N. 2011, Journal of Contemporary Physics, 52, 5, 403
Horne, J. H. & Baliunas, S. L. 1986, ApJ, 302, 757
Mayor, M. & Queloz, D. 1995, Nature, 378, 355
Marsaglia, G. & Zaman, A. 1987, Florida State University Report: FSU-SCRI-87-50
Press W. H. et al. 2007, *Numerical recipes in C* (Cambridge University Press, Camdbridge, UK)
Quirrenbach, A. et al. 2012, SPIE, 8446, E0R
Rivera, E. J. et al. 2005, ApJ, 634, 625
Scargle, J. D. 1982, ApJ, 263, 835
Schulz, M. & Mudelsee, M. 2002, Computer & Geosciences, 28, 421
Steffen, J. H. & Agol, E. 2005, MNRAS, 364, L96
Vogt, S. S. et al. 2010, ApJ, 723, 954
Vogt, S. S., Butler, R. P. & Haghighipour, N. 2012, AN, 333, 561
Wilks D. S., 2006, *Statistical methods in the atmospheric sciences* (ed. Elsevier)
Wright, J. T. & Howard, A. W. 2009, ApJS, 182, 205

# Apéndice B

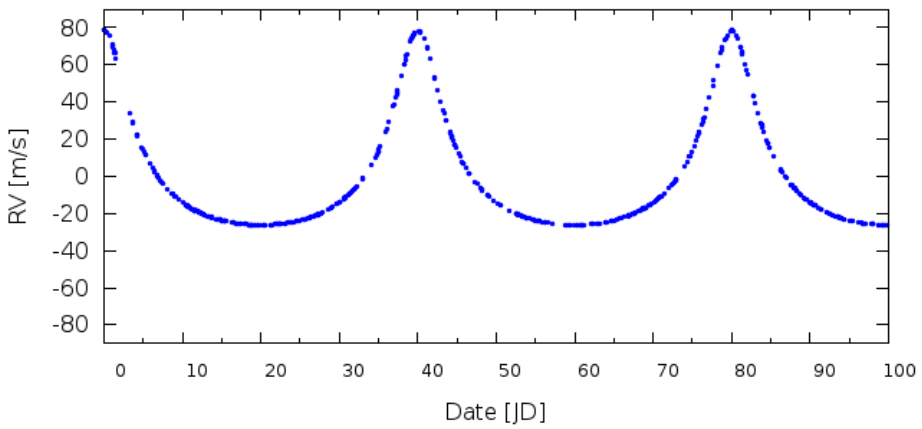
# Figuras




(a) Curva de velocidad radial con $e = 0{,}0$.

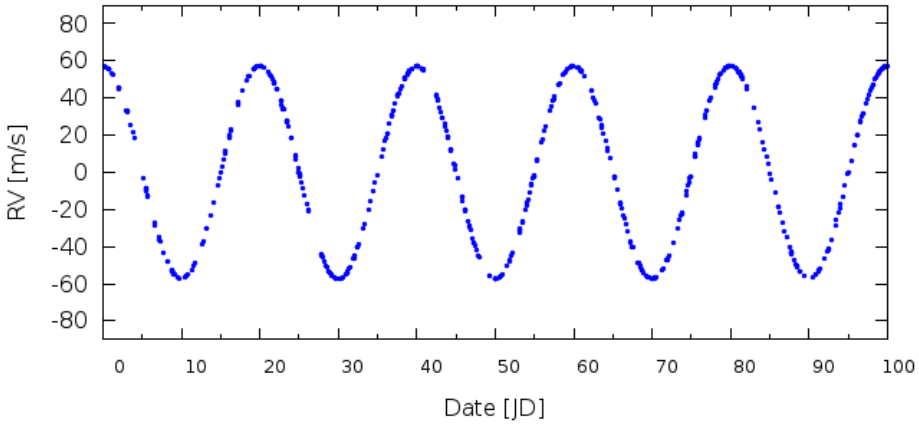



(b) Curva de velocidad radial con $e = 0{,}3$.

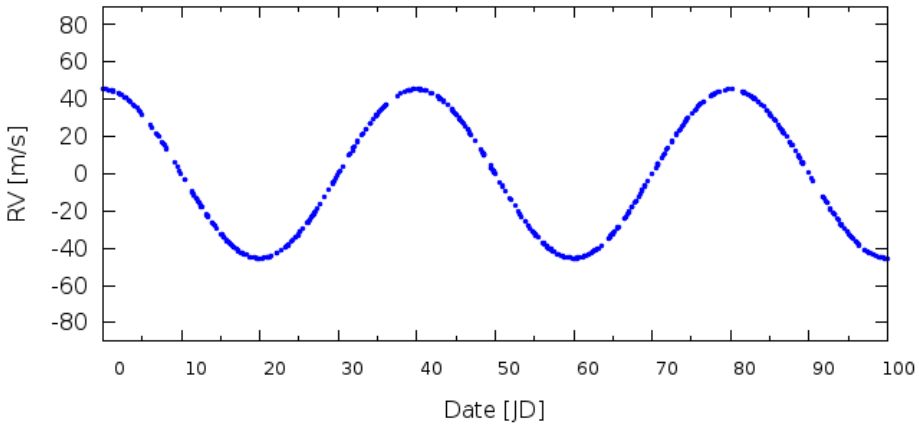



(c) Curva de velocidad radial con $e = 0{,}5$.

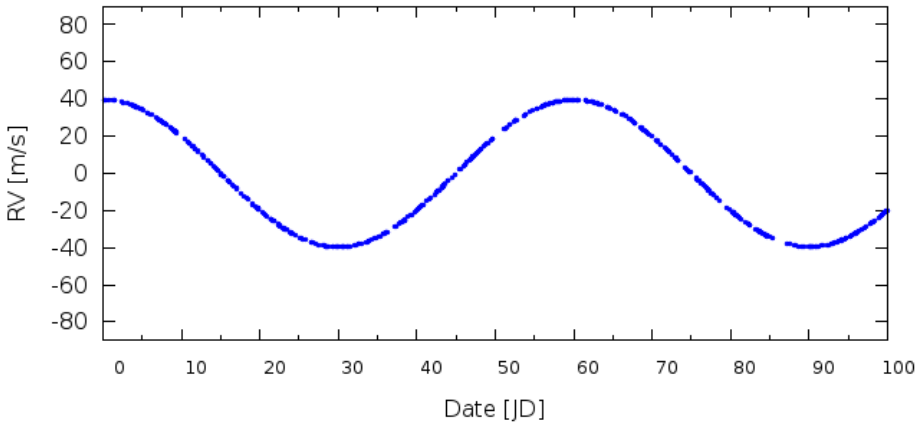
Figura B.1: Curvas de velocidad radial con distintas excentricidades.




(a) Curva de velocidad radial con $P = 20$ días.

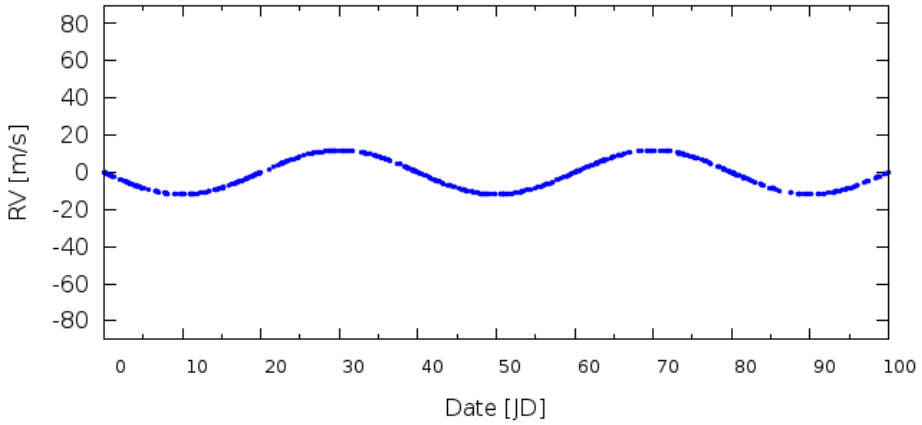



(b) Curva de velocidad radial con $P = 40$ días.

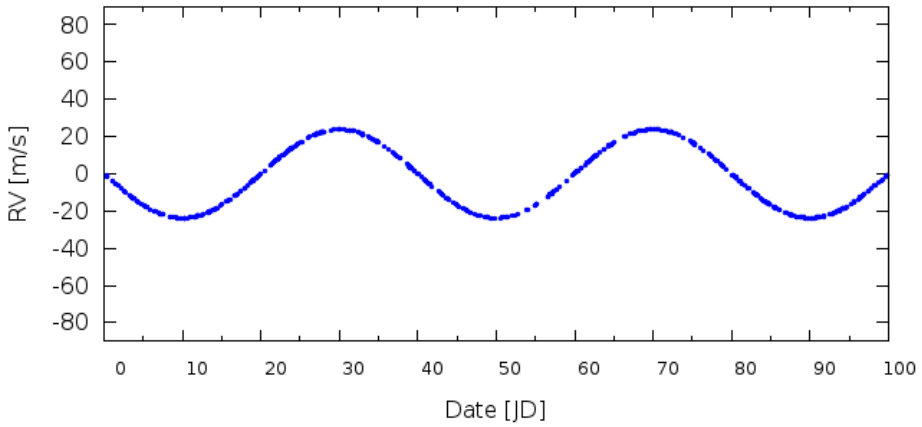



(c) Curva de velocidad radial con $P = 60$ días.

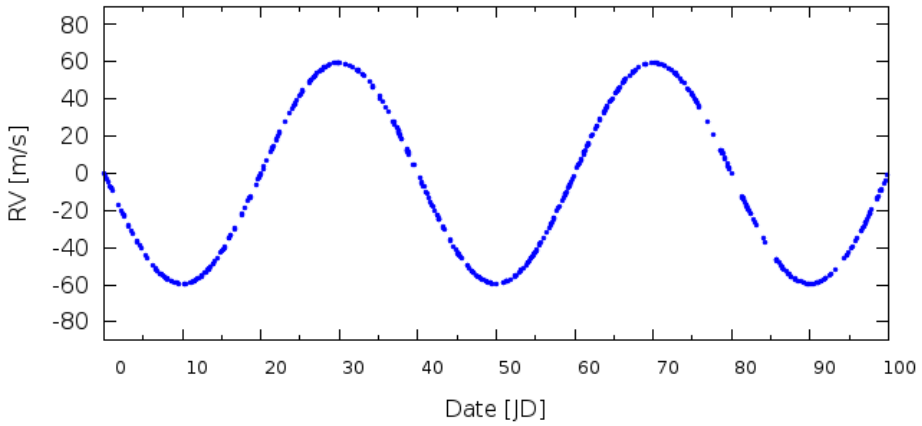
Figura B.2: Curvas de velocidad radial con distintos períodos.




(a) Curva de velocidad radial con $K_1 = 11.9\ \mathrm{m\,s^{-1}}$.

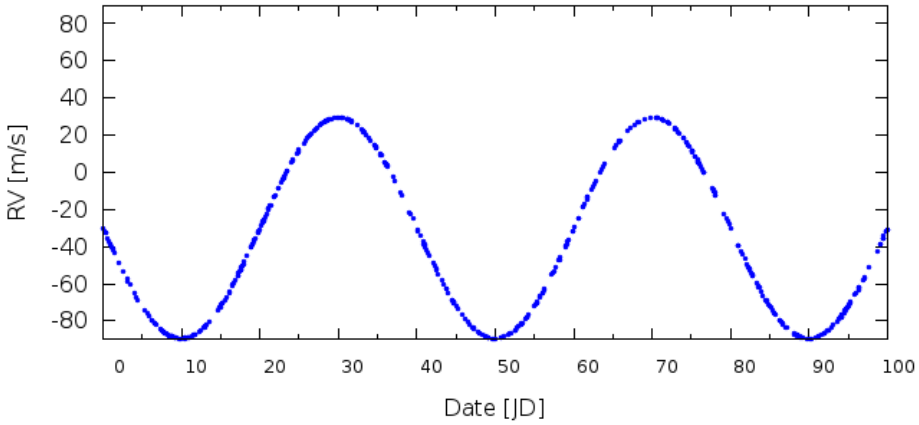



(b) Curva de velocidad radial con $K_1 = 23.8\ \mathrm{m\,s^{-1}}$.

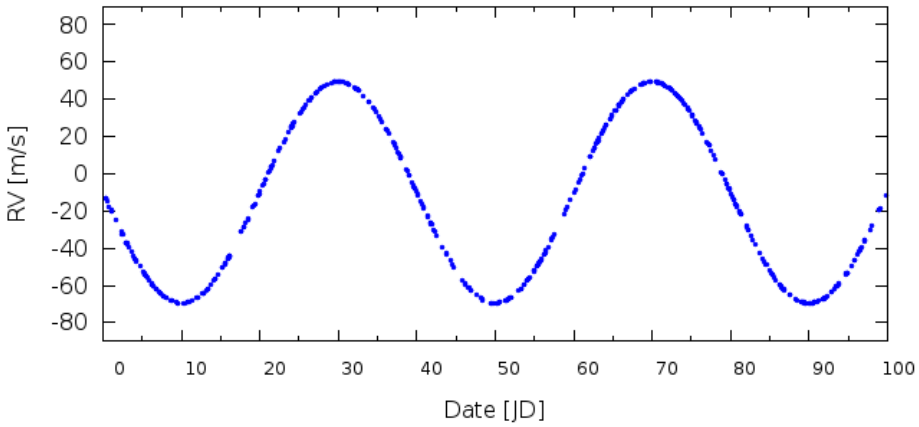



(c) Curva de velocidad radial con $K_1 = 59.4\ \mathrm{m\,s^{-1}}$.

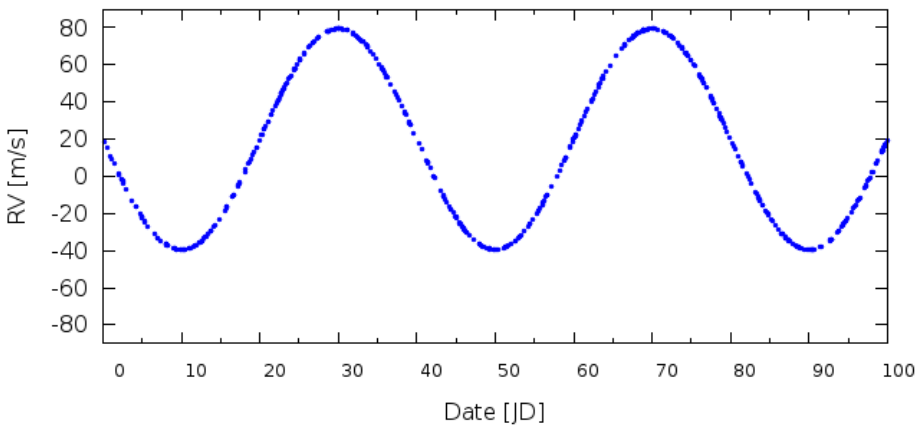
Figura B.3: Curvas de velocidad radial con distintas semiamplitudes.




(a) Curva de velocidad radial con $V_0$ = -0.03 $\mathrm{km\,s^{-1}}$.

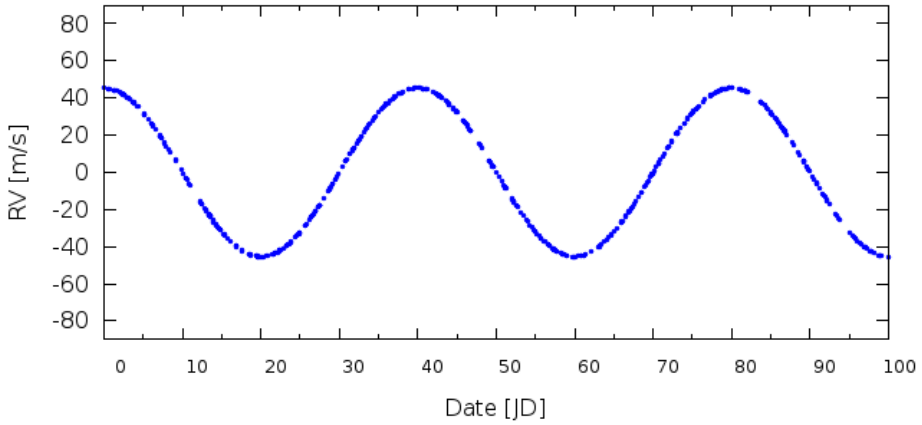



(b) Curva de velocidad radial con $V_0$ = -0.01 $\mathrm{km\,s^{-1}}$.

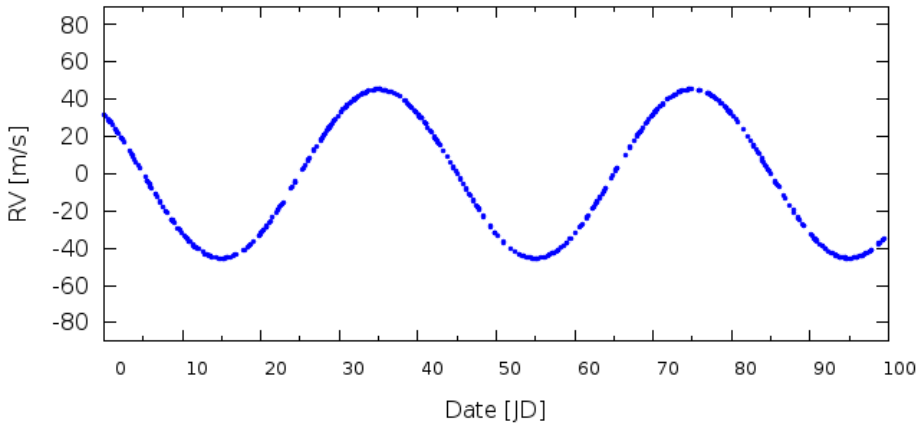



(c) Curva de velocidad radial con $V_0$ = 0.02 $\mathrm{km\,s^{-1}}$.

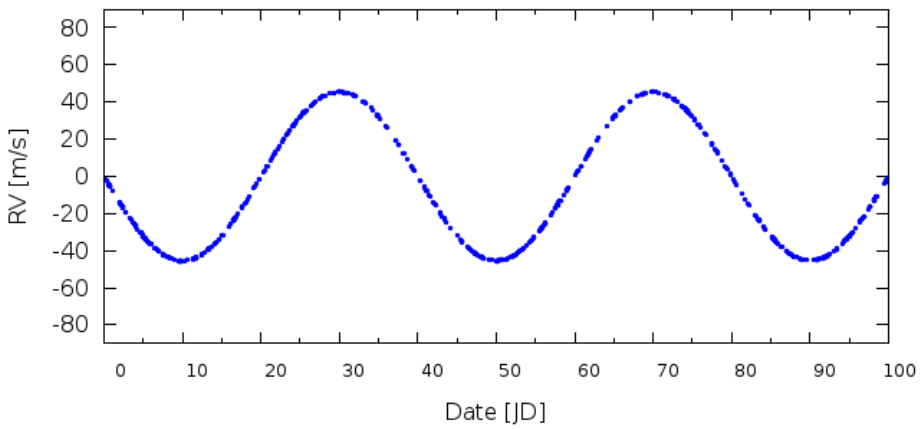
Figura B.4: Curvas de velocidad radial con distintas velocidades sistémicas.




(a) Curva de velocidad radial con $\omega = 0$ deg.




(b) Curva de velocidad radial con $\omega = 45$ deg.




(c) Curva de velocidad radial con $\omega = 90$ deg.

Figura B.5: Curvas de velocidad radial con distintas velocidades sistémicas.

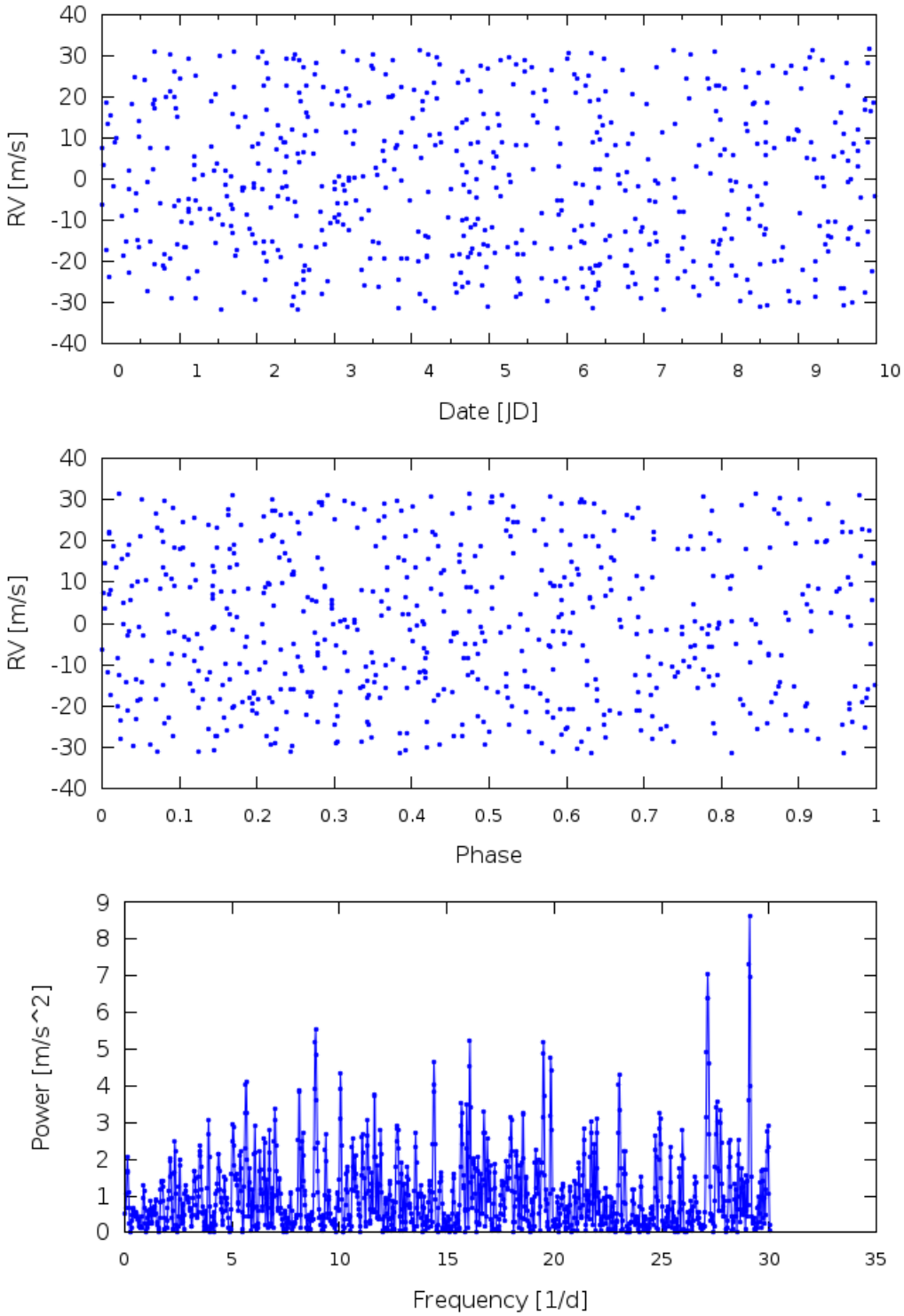



Figura B.6: Señal de ruido no gaussiano (panel superior) y su espectro de potencias (panel inferior).

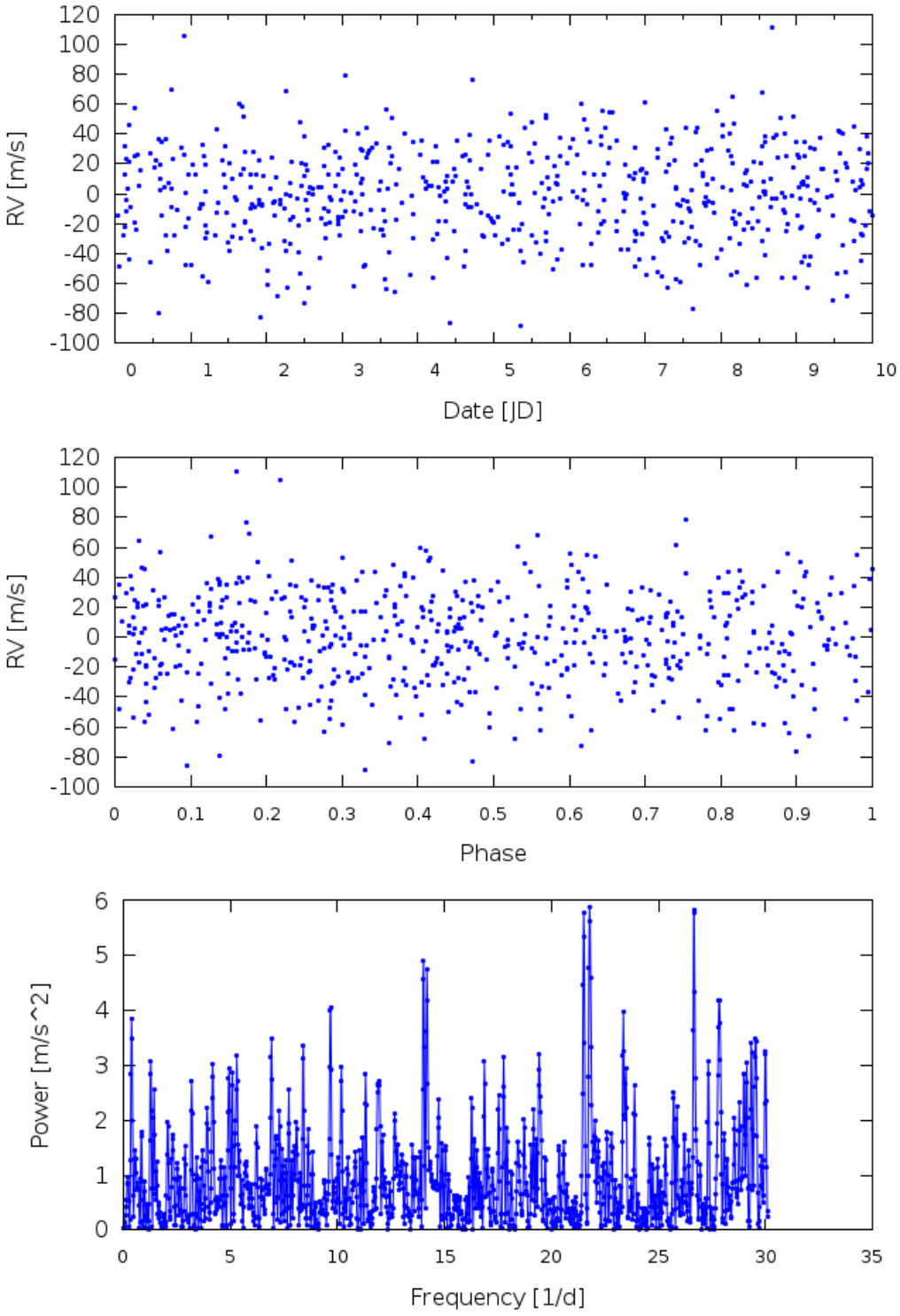



Figura B.7: Señal de ruido blanco gaussiano (panel superior) y su espectro de potencias (panel inferior).

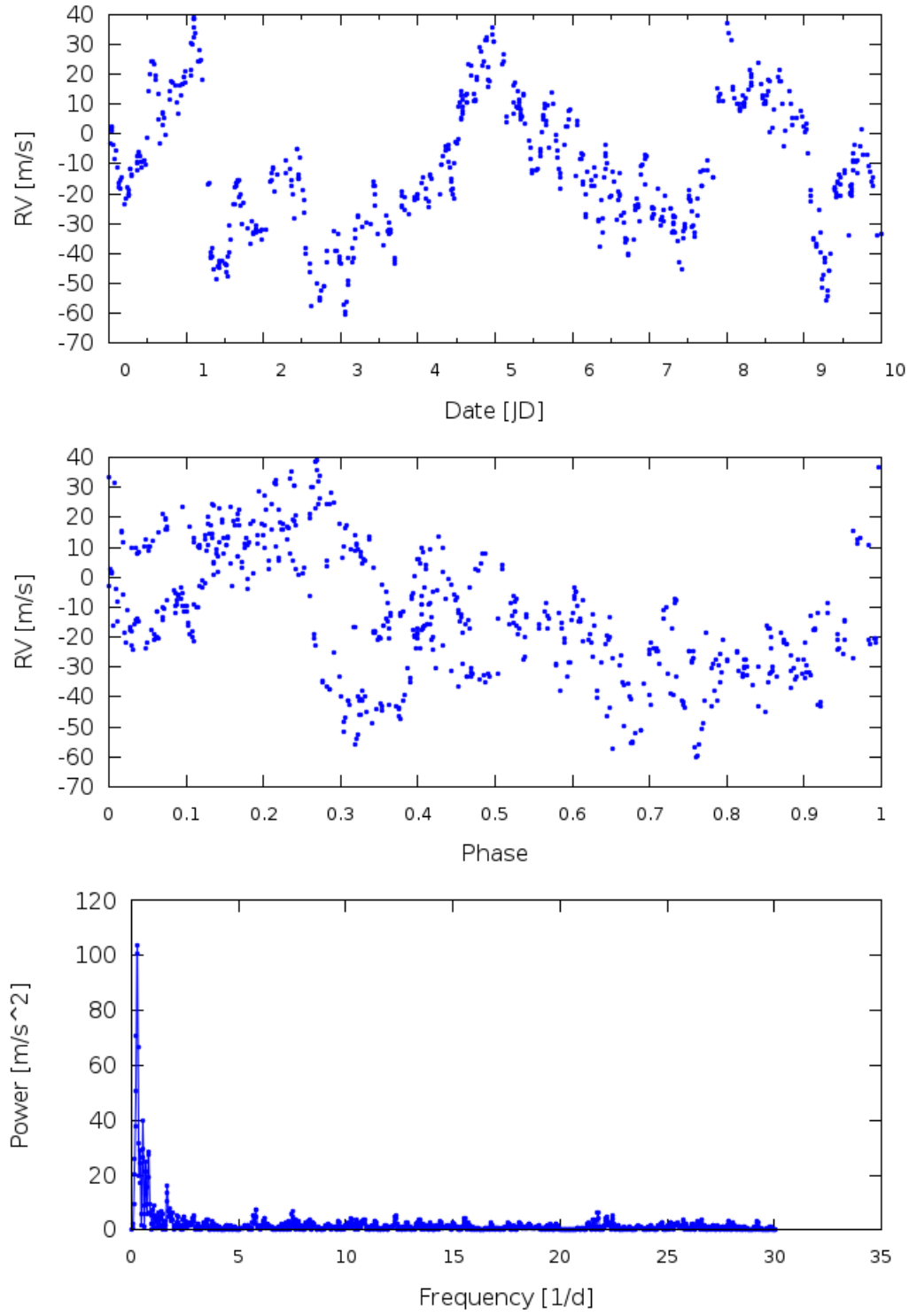
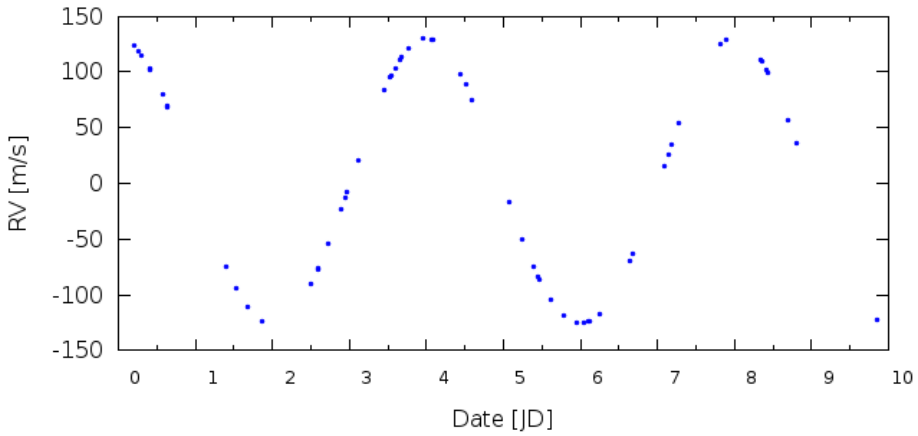
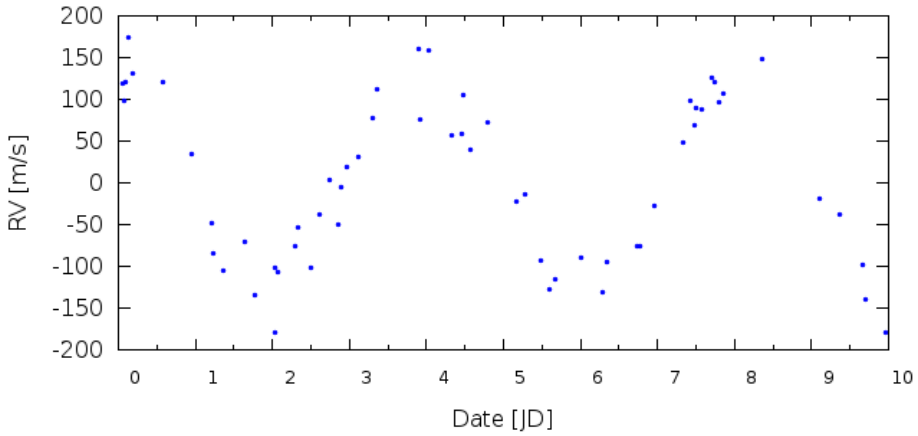


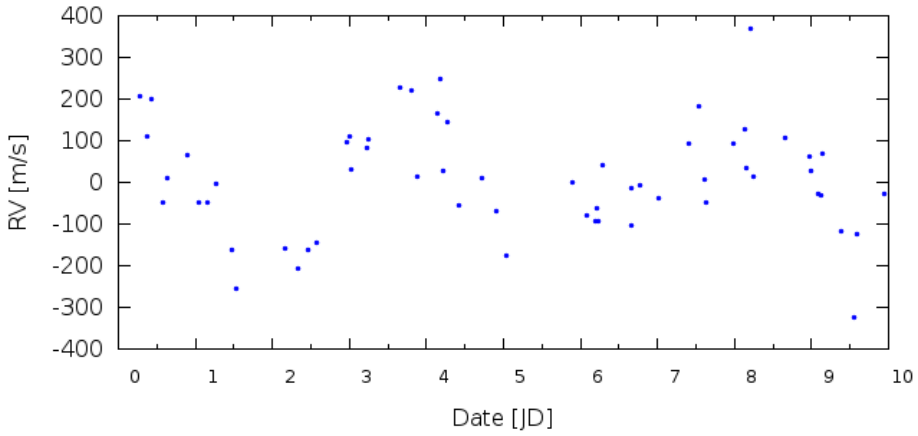

Figura B.8: Señal de ruido rojo (panel superior) y su espectro de potencias (panel inferior).




(a) Curva sin ruido.

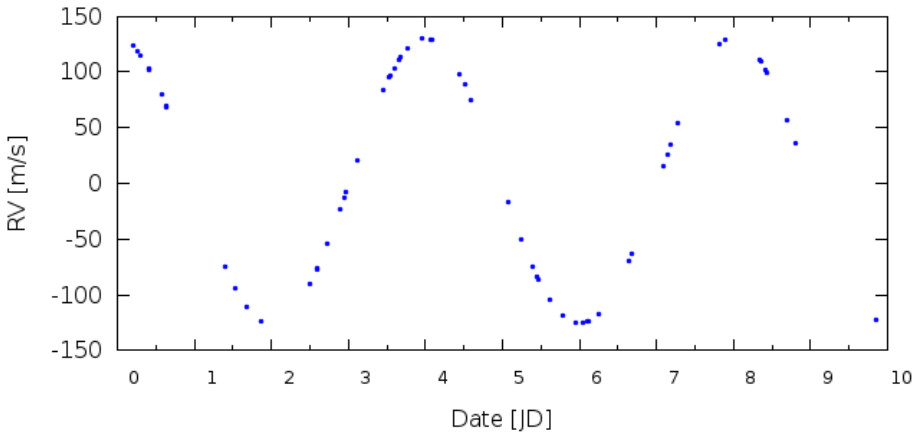



(b) Curva con ruido blanco gaussiano de amplitud 30 m/s.

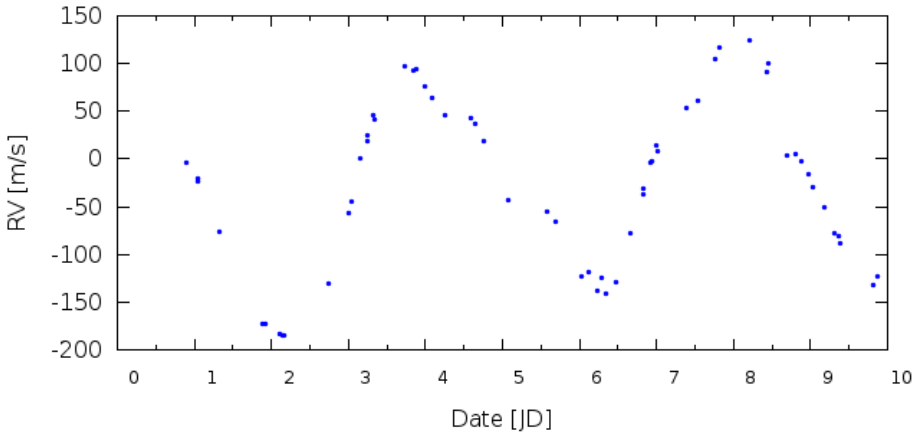



(c) Curva con ruido blanco gaussiano de amplitud 100 m/s.

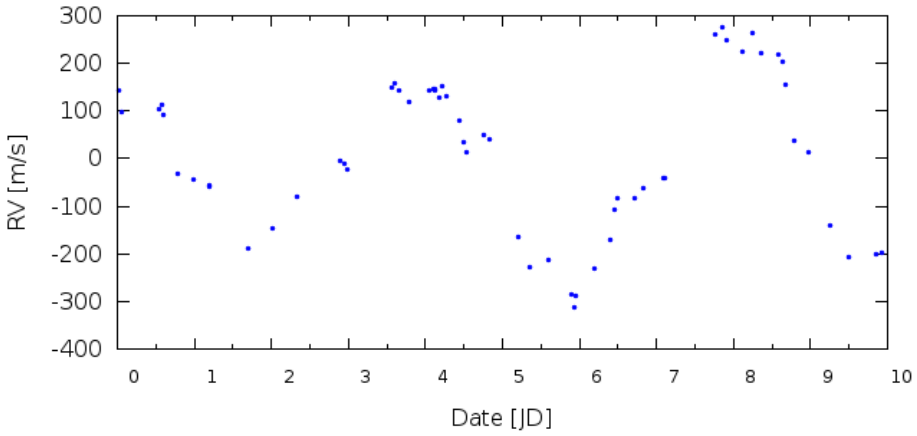
Figura B.9: Curvas de velocidad radial con un ruido blanco para un Júpiter caliente.




(a) Curva sin ruido.




(b) Curva con ruido rojo de 30 m/s.




(c) Curva con ruido rojo de 100 m/s.

Figura B.10: Curvas de velocidad radial con ruido rojo para un Júpiter caliente.

# Apéndice C

# Código fuente

## file_read.c

```
#include "rn.h"

/*****************************************************************************
 * FUNCIÕN: DataRead
 *****************************************************************************
 * Función para la lectura de archivos de texplo plano con dos o tres columnas
 *****************************************************************************
 * Argumentos:
 * file archivo a leer
 *****************************************************************************
 * Devuelve una estructura con las columnas del archivo en arrays
 *****************************************************************************/
serieTemporal DataRead (char file[])
{
    //***************************************************************************
    // Lectura archivo con la curva rv
    //***************************************************************************
    FILE *ifile = fopen(file, "r");
    if (ifile == 0)
    {
    fprintf (stdout,"-------------------------------------------------\n");
    fprintf (stdout,"  No existe el archivo '%s'.\n", file);
    fprintf (stdout,"-------------------------------------------------\n");
    exit (-1);
    }
    //***************************************************************************
    // Conteo del # de lineas del archivo
    //***************************************************************************
    int n, i, j;
    int line = 0;
    char buf [INDEXMAX];
    while ((fgets (buf, sizeof(buf), ifile)) != NULL)       line++;
    n = line;
    rewind (ifile);

    serieTemporal serie;
    serie.size = n;

    i = 0;
    char car;
    int nCol = 1;
    //***************************************************************************
    // Conteo del # de columnas
    //***************************************************************************
    while ((car = fgetc(ifile)) != '\n')
    {
    if (car == '\t') nCol++;
    }
    rewind (ifile);
    //***************************************************************************
    // Creación de los arrays dentro de la estructura de la serie temporal
    //***************************************************************************
    if (nCol == 1)
    {
        while ((car = fgetc(ifile)) != '\n')
```

```
{
if (car == ' ') nCol++;
}
rewind (ifile);
    if (nCol == 2)
    {
    while(!feof(ifile))
    {
    fscanf (ifile,"%lf %lf", &serie.t[i], &serie.rv[i]);
    i++;
    }
    }
    if (nCol == 3)
    {
    while(!feof(ifile))
    {
    fscanf (ifile,"%lf %lf %f", &serie.t[i], &serie.rv[i], &serie.other[i]);
    i++;
    }
    }
    }
    else if (nCol == 2)
    {
    while(!feof(ifile))
    {
    fscanf (ifile,"%lf\t%lf", &serie.t[i], &serie.rv[i]);
    i++;
    }
    }
    else if (nCol == 3)
    {
    while(!feof(ifile))
    {
    fscanf (ifile,"%lf\t%lf\t%f", &serie.t[i], &serie.rv[i], &serie.other[i]);
    i++;
    }
    }
    else
    {
    fprintf (stdout,"-------------------------------------------------------\n");
    fprintf (stdout,"  Las separación de las columnas debe ser TAB o un espacio.\n\n");
    fprintf (stdout,"-------------------------------------------------------\n");
    exit (-1);
}
    fclose (ifile);

    return serie;
}


/**************************************************************************
 * FUNCIÓN:
 **************************************************************************
 * Función para leer un archivo de texto plano y mostrarlo en pantalla
 **************************************************************************
 * Argumentos:
 * file nombre del archivo a imprimir en pantalla
 **************************************************************************
 * Devuelve una cadena de caracteres con el contenido del archivo de texto
 **************************************************************************/
char* ReadFile(char *filename)
{
   char *buffer = NULL;
   int string_size,read_size;
   FILE *ifile = fopen(filename,"r");

   if (ifile)
   {
fseek(ifile,0,SEEK_END); // buscar el ultimo byte del archivo
string_size = ftell (ifile); // offset del primer al ultimo byte = tamaño del archivo
rewind(ifile); // volver al principio del archivo

buffer = (char*) malloc (sizeof(char) * (string_size + 1) ); // alojar un string dinamicamente para el texto
read_size = fread(buffer,sizeof(char),string_size,ifile); // leer todo (una operacion)

buffer[string_size+1] = '\0'; // ya que fread no lo hace,colocamos un \0 al final del buffer,ahora es un string

if (string_size != read_size) // error, liberar la memoria, y colocar el buffer como NULL
{
free(buffer);
buffer = NULL;
}
    }
```

```
    return buffer;
}
```

## main.c

```
#include "rn.h"

int main (int argc, char *argv[])
{
fprintf (stdout,"\n\n");
    //*************************************************************************
// Opción para procesar con el LS un archivo con curva RV
//*************************************************************************
    if (strcmp(argv[1],"--file") == 0 || strcmp(argv[1],"-f") == 0)
{
        fprintf (stdout, "-------------------------------------------------\n");
        fprintf (stdout, "  REDNOISE v%1.1f : \n", VERSION);
        fprintf (stdout, "-------------------------------------------------\n");
    double Period = Powerspectrum (argv[2], 0);
        PrintGnuplot (1, Period, 0, argv[2], argv[2]);
        PrintGnuplot (2, Period, 0, argv[2], argv[2]);
        fprintf (stdout,"\n");
        return 0;
}
    //*************************************************************************
// Opción para mostrar la ayuda en pantalla
//*************************************************************************
    if (strcmp(argv[1],"--help") == 0 || strcmp(argv[1],"-?") == 0)
{
fprintf (stdout,"\n                              RedNoise v%1.1f \n\n", VERSION);
        char *string = ReadFile("README.txt");                          // Lee el texto del archivo README.txt
if (string)
{
puts(string);
free(string);
}
return 0;
}
    fprintf (stdout, "-------------------------------------------------\n");
fprintf (stdout, "  REDNOISE v%1.1f : \n", VERSION);
    fprintf (stdout, "-------------------------------------------------\n");

    //*************************************************************************
// Opción para generar una sinusoide con ruido y obetener su espectro
//*************************************************************************
    if (strcmp(argv[1],"--sin") == 0 && argc == 11 || argc == 12)
    {
        double t_0 = atof(argv[3]);
        double t_f = atof(argv[4]);
        float P = atof(argv[5]);
        int N = atoi(argv[6]);
        float K_1 = atof(argv[7]);
        float gamma = atof(argv[8]);
        float phi = atof(argv[9]);
        float noise = atof(argv[10]);

    if (atoi(argv[2]) == 1 )  fprintf (stdout,"  Sinusoide con Ruido no Gaussiano: \n\n");
    else if (atoi(argv[2]) == 2 )  fprintf (stdout,"  Sinusoide con Ruido Blanco Gaussiano: \n\n");
    else if (atoi(argv[2]) == 3 )  fprintf (stdout,"  Sinusoide con Ruido Rojo: \n\n");
        else
        {
            fprintf (stdout, "  Parámetros insuficientes o incorrectos. \n  Para ver la ayuda ejecuta: ./rn --help \n");
            fprintf (stdout, "-------------------------------------------------\n");
            return 0;
        }
        fprintf (stdout,"Parámetros : \n\n");
        fprintf (stdout,"T inicial    = %.0f JD \n", t_0);
        fprintf (stdout,"T final      = %.0f JD \n", t_f);
        fprintf (stdout,"N puntos     = %d \n", N);
        fprintf (stdout,"P            = %.2f días \n", P);
        fprintf (stdout,"K_1          = %.2f m/s \n", K_1);
        fprintf (stdout,"gamma        = %.2f km/s \n", gamma);
        fprintf (stdout,"phi          = %.1f grados \n", phi);
        fprintf (stdout,"noise        = %.3f m/s \n", sqrt(noise));
    if (atoi(argv[2]) == 1 )  SinCurve (t_0, t_f, P, N, K_1, gamma, phi, noise, 0.0, atoi(argv[2]));
    else if (atoi(argv[2]) == 2 )  SinCurve (t_0, t_f, P, N, K_1, gamma, phi, noise, 0.0, atoi(argv[2]));
        else if (atoi(argv[2]) == 3)
        {
            float tau = atof(argv[11]);
            fprintf (stdout,"  tau          = %.2f \n", tau);
            SinCurve (t_0, t_f, P, N, K_1, gamma, phi, noise, tau, atoi(argv[2]));
        }
```

```
        Powerspectrum ("data_curve.dat", 0);
    Powerspectrum_lomb (1);
        PrintGnuplot (1, atof(argv[5]), 0, "data_curve.dat", "data_curve.dat"); // PrintGnuplot hace el phase plot con
        PrintGnuplot (2, atof(argv[5]), 0, "data_curve.dat", "data_curve.dat"); // el periodo de modelado de la sinusoide
        fprintf (stdout,"\n");
}
    //*************************************************************************
// Opción para modelar una curva RV con ruido, y obtener su espectro
//*************************************************************************
else if (strcmp(argv[1],"--rv") == 0 && argc == 13 || argc == 14)
{
        double t_0 = atof(argv[3]);
        double t_f = atof(argv[4]);
        float P = atof(argv[5]);
        int N = atoi(argv[6]);
        float gamma = atof(argv[7]);
        float e = atof(argv[8]);
        float m2sinI = atof(argv[9]);
        float m1 = atof(argv[10]);
        float w = atof(argv[11]);
        float noise = atof(argv[12]);

        if (atoi(argv[2]) == 1 )  fprintf (stdout,"  RV con Ruido no Gaussiano: \n\n");
    else if (atoi(argv[2]) == 2 )  fprintf (stdout,"  RV con Ruido Blanco Gaussiano: \n\n");
    else if (atoi(argv[2]) == 3 )  fprintf (stdout,"  RV con Ruido Rojo: \n\n");
        else
        {
            fprintf (stdout, "  Parámetros insuficientes o incorrectos. \n  Para ver la ayuda ejecuta: ./rn --help \n");
            fprintf (stdout, "-------------------------------------------------\n");
            return 0;
        }
        fprintf (stdout,"Parámetros : \n\n");
        fprintf (stdout,"T inicial    = %.0f JD \n", t_0);
        fprintf (stdout,"T final      = %.0f JD \n", t_f);
        fprintf (stdout,"N puntos     = %d \n", N);
        fprintf (stdout,"P            = %.2f días \n", P);
        fprintf (stdout,"gamma        = %.2f km/s \n", gamma);
        fprintf (stdout,"e            = %1.3f \n", e);
        fprintf (stdout,"m2*sin(I)    = %.4f m_jup \n", m2sinI);
        fprintf (stdout,"m1           = %.4f m_sol \n", m1);
        fprintf (stdout,"w            = %.1f grados\n", w);
        fprintf (stdout,"noise        = %.3f m/s \n", sqrt(noise));

        if (atoi(argv[2]) == 1 )  RvCurve (t_0, t_f, P, N, gamma, e, m2sinI, m1, w, noise, 0.0, atoi(argv[2]));
        else if (atoi(argv[2]) == 2 )  RvCurve (t_0, t_f, P, N, gamma, e, m2sinI, m1, w, noise, 0.0, atoi(argv[2]));
        else if (atoi(argv[2]) == 3 )
        {
            float tau = atof(argv[13]);
            fprintf (stdout,"tau          = %.2f \n", tau);
            RvCurve (t_0, t_f, P, N, gamma, e, m2sinI, m1, w, noise, tau, atoi(argv[2]));
        }
        Powerspectrum ("data_curve.dat", 0);
    Powerspectrum_lomb (1);
        PrintGnuplot (1, atof(argv[5]), 0, "data_curve.dat", "data_curve.dat"); // PrintGnuplot hace el phase plot con
        PrintGnuplot (2, atof(argv[5]), 0, "data_curve.dat", "data_curve.dat"); // el periodo de modelado de la RV
        fprintf (stdout,"\n");
}
    //*************************************************************************
// Opción de llamada a EXOFIT para ajuste de los parametros orbiales de la RV
//*************************************************************************
else if (strcmp(argv[1],"--rvfit") == 0)
    {
        fprintf (stdout,"  Para el ajuste de los parametros orbiales de la RV: \n");
        RvCurveFit();
        fprintf (stdout, "-------------------------------------------------\n");
    }
    //*************************************************************************
// Opción de extración del ruido (rojo)  (RV con ruido - RV ajuste)
//*************************************************************************
else if (strcmp(argv[1],"--rnext") == 0 && argc == 8)
    {
        float P = atof(argv[2]);
        float K_1 = atof(argv[3]);
        float gamma = atof(argv[4]);
        float e = atof(argv[5]);
        float w = atof(argv[6]);
        int ventana = atoi(argv[7]);
        fprintf (stdout,"  Extracción del Ruido y su espectro: \n");
        // Copia de la RV modelada que fué ajustada en EXOFIT hasta la carpeta raíz
//system ("cp 'data/fit/data_curve_mod.dat' data_curve_mod.dat");
        double Period = Powerspectrum ("data_curve_mod.dat", 0);              // data_curve_mod.dat : RV con ruido modelada
    Powerspectrum_lomb (3);
RnExtract (P, K_1, gamma, e, w, ventana);
```

```
        PrintGnuplot (3, Period, atof(argv[2]), "data_curve_mod.dat", "data_curve_noisig.dat");
        PrintGnuplot (4, Period, atof(argv[2]), "data_curve_mod.dat", "data_curve_noisig.dat");
        fprintf (stdout, "-------------------------------------------------\n");
    }
    //*************************************************************************
// Opción de llamada a MATLAB para ajuste del espectro del ruido
//*************************************************************************
else if (strcmp(argv[1],"--rnfit") == 0 && argc == 2)
    {
        fprintf (stdout,"  Ajuste en MATLAB de los parámetros del espectro \n  del Ruido Rojo: \n");
        fprintf (stdout, "-------------------------------------------------\n");
        Powerspectrum_lomb (2); // Repetimos el espectro del ruido extraido
        RnFit();
        fprintf (stdout, "-------------------------------------------------\n");
    }
    //*************************************************************************
// Opción para el test de significación de los picos de la RV con respecto
    // al espectro teórico del ruido rojo
//*************************************************************************
else if (strcmp(argv[1],"--rntest") == 0 && argc == 6)
    {
        float Rho = atof(argv[2]);
        float A = atof(argv[3]);
        float fNyq = atof(argv[4]);
        float alpha = atof(argv[5]);
        fprintf (stdout,"  Plot con el Test de significación para \n  alpha = %.3f: \n",  alpha);
        fprintf (stdout, "-------------------------------------------------\n\n");
        Powerspectrum_lomb (3); // Repetimos el espectro de la señal modelada
        //RnTest (Rho, A, fNyq, alpha); // con ruido (data_curve_mod.dat)
        RnTestMultiple (Rho, A, fNyq, alpha);
        PrintGnuplotRn (1, alpha);
        PrintGnuplotRn (2, alpha);
        fprintf (stdout, "-------------------------------------------------\n");
    }

    else if (strcmp(argv[1],"--gaustest") == 0 && argc == 4)
{
GausTest (atof(argv[2]), atof(argv[3]));
        fprintf (stdout, "-------------------------------------------------\n");
}
else if (strcmp(argv[1],"--powrv") == 0 && argc == 2)
{
system ("octave matlab_espectro_rv.m");
}
else
{
fprintf (stdout, "  Parámetros insuficientes o incorrectos. \n  Para ver la ayuda ejecuta: ./rn --help \n");
        fprintf (stdout, "-------------------------------------------------\n");
}

return 0;
}
```

## makefile

```
.SUFFIXES: .o .c
.c.o:
$(CC) -c $(CFLAGS) $<

CC = gcc
CFLAGS = -lm
OBJ = main.o sinusoid_curve.o rv_curve.o powerspectrum.o randomlib.o print_gnuplot.o noise.o rvfit_rnext_rnfit.o file_read.o varlib.o

rn : $(OBJ)
$(CC) $(OBJ) -o rn $(CFLAGS)

varlib.o : rn.h
file_read.o : rn.h
rvfit_rnext_rnfit.o : rn.h
noise.o : rn.h
sinusoid_curve.o : rn.h
rv_curve.o : rn.h
powerspectrum.o : rn.h
main.o : rn.h
randomlib.o : rn.h
print_gnuplot.o : rn.h

.PHONY : clean
clean:
rm -f $(OBJ)
```

## matlab_espectro_lomb.m

```
function [f,P,prob] = matlab_espectro_lomb(t,h,ofac,hifac)
% LOMB(T,H,OFAC,HIFAC) computes the Lomb normalized periodogram (spectral
% power as a function of frequency) of a sequence of N data points H,
% sampled at times T, which are not necessarily evenly spaced. T and H must
% be vectors of equal size. The routine will calculate the spectral power
% for an increasing sequence of frequencies (in reciprocal units of the
% time array T) up to HIFAC times the average Nyquist frequency, with an
% oversampling factor of OFAC (typically >= 4).
%
% The returned values are arrays of frequencies considered (f), the
% associated spectral power (P) and estimated significance of the power
% values (prob).  Note: the significance returned is the false alarm
% probability of the null hypothesis, i.e. that the data is composed of
% independent gaussian random variables.  Low probability values indicate a
% high degree of significance in the associated periodic signal.
%
% Although this implementation is based on that described in Press,
% Teukolsky, et al. Numerical Recipes  In C, section 13.8, rather than using
% trigonometric rercurrences, this takes advantage of MATALB's array
% operators to calculate the exact spectral power as defined in equation
% 13.8.4 on page 577.  This may cause memory issues for large data sets and
% frequency ranges.
%
% Example
%    [f,P,prob] = lomb(t,h,4,1);
%    plot(f,P)
%    [Pmax,jmax] = max(P)
%    disp(['Most significant period is ',num2str(1/f(jmax)),...
%         ' with FAP of ',num2str(prob(jmax))])
%
% Written by Dmitry Savransky 21 May 2008

%sample length and time span
N = length(h);
T = max(t) - min(t);

%mean and variance
mu = mean(h);
s2 = var(h);

%calculate sampling frequencies
f = (1/(T*ofac):1/(T*ofac):hifac*N/(2*T)).';

%angular frequencies and constant offsets
w = 2*pi*f;
tau = atan2(sum(sin(2*w*t.'),2),sum(cos(2*w*t.'),2))./(2*w);

%spectral power
cterm = cos(w*t.' - repmat(w.*tau,1,length(t)));
sterm = sin(w*t.' - repmat(w.*tau,1,length(t)));
P = (sum(cterm*diag(h-mu),2).^2./sum(cterm.^2,2) + ...
     sum(sterm*diag(h-mu),2).^2./sum(sterm.^2,2))/(2*s2);

%estimate of the number of independent frequencies
M=2*length(f)/ofac;

%statistical significane of power
prob = M*exp(-P);
inds = prob > 0.01;
prob(inds) = 1-(1-exp(-P(inds))).^M;
```

## matlab_espectro_rn.m

```
ts = load('data_curve_noisig.dat');
t = ts(:,1);
rn = ts(:,2);
N = length(t);
Dt_ave = (t(N)-t(1))/N;
tau = 20

% Llamada a función lomb-scargle
hifac=1;
[frec,ps_rn,prob]=matlab_espectro_lomb(t,rn,4,hifac);
frec=frec';
ps_rn=ps_rn';
% Encuentra el periodo max
[PSmax,ind_PSmax] = max (ps_rn);
F_PSmax = frec(ind_PSmax);
P_lomb2 = 1/F_PSmax
%-----------------
fNyq=1/Dt_ave/2
rho_teor=exp(-Dt_ave/tau)
```

```
F = [frec',ps_rn'];
%save ('data_power_noisig_fnyq.dat','F','-ascii');
dlmwrite('data_power_noisig_fnyq.dat',F, '\t');
```

## matlab_espectro_rv.m

```
ts = load('data_curve.dat');
t = ts(:,1);
rn = ts(:,2);
N = length(t);
Dt_ave = (t(N)-t(1))/N;
%tau = 5;

% Llamada a función lomb-scargle
hifac=1;
[frec,ps,prob]=matlab_espectro_lomb(t,rn,4,hifac);
frec=frec';
ps=ps';
% Encuentra el periodo max
[PSmax,ind_PSmax] = max (ps);
F_PSmax = frec(ind_PSmax);
P_lomb2 = 1/F_PSmax
fNyq=1/Dt_ave/2
F = [frec',ps'];
%save ('data_power.dat','F','-ascii');
dlmwrite('data_power.dat',F, '\t');
```

## matlab_espectro_rvmod.m

```
ts = load('data_curve_mod.dat');
t = ts(:,1);
rn = ts(:,2);
N = length(t);
Dt_ave = (t(N)-t(1))/N;
%tau = 5;

% Llamada a función lomb-scargle
hifac=1;
[frec,ps,prob]=matlab_espectro_lomb(t,rn,4,hifac);
frec=frec';
ps=ps';
% Encuentra el periodo max
[PSmax,ind_PSmax] = max (ps);
F_PSmax = frec(ind_PSmax);
P_lomb2 = 1/F_PSmax
fNyq=1/Dt_ave/2
F = [frec',ps'];
%save ('data_power.dat','M','-ascii');
dlmwrite('data_power.dat',F, '\t');
```

## matlab_fit.m

```
M=load('data_power_noisig_fnyq.dat');
f=M(:,1);
p=M(:,2);
deltaTave=0.198729;
Tau=0.2;

matlab_fit_function(f,p)
Rho_ruido = exp (-deltaTave/Tau)

exit
```

## matlab_fit_function.m

```
function [fitresult, gof] = matlab_fit_function(f, p)

%% Fit: 'Red Noise'.
[xData, yData] = prepareCurveData( f, p );

% FITTYPE (ecuación modelo) y OPCIONES
ft = fittype( 'A / (1 + Rho^2 - 2 * Rho * cos(pi * x / 2.5160 ))', 'independent', 'x', 'dependent', 'y' );
opts = fitoptions( ft );
opts.Display = 'Off';
opts.Lower = [-Inf -Inf];
opts.StartPoint = [0.849129305868777 0.933993247757551];
opts.Upper = [Inf Inf];
```

```
% FIT
[fitresult, gof] = fit( xData, yData, ft, opts );

% PLOT
f = figure( 'Name', 'Red Noise' );
h = plot( fitresult, xData, yData );
legend( h, 'p vs. f', 'Red Noise', 'Location', 'NorthEast' );
% Label axes
xlabel( 'f' );
ylabel( 'p' );
print(f,'plot2.png','-dpng')
```

## noise.c

```
#include "rn.h"

/*****************************************************************************
 * FUNCIÕN: Noise
 *****************************************************************************
 * Función para agregar ruido a la sinusoide o a la RV modelada. El ruido puede
 * ser de 3 tipos: ruido no gaussiano, ruido blanco gaussiano y ruido rojo
 *****************************************************************************
 * Argumentos:
 * arg argumento switch para controlar el flujo de la función y generar
 *  alguno de los tres tipos de ruidos y sumarlos a la señal
 * N número de puntos de la serie temporal de la curva
 * noise amplitud del ruido
 * tau parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *  se usa en el caso de querer generar ruido rojo
 * *t array con los tiempos de la serie temporal
 * *rv array con las medidas de RV de la curva o con la sinusoide
 *****************************************************************************
 * Devuelve 0 en caso de finalizar correctamente
 *****************************************************************************/
int Noise (int arg, int N, float noise, float tau, double *t, double *rv)
{
    int i, j, k;

    FILE *ofile = fopen("data_curve.dat", "w");

    // Inicializacion de la semilla y los aleatorios
    double seed = time(NULL) % 30000;
    RandomInitialise (seed, seed+100);

    //***************************************************************************
    // Serie (sinusoide o RV) Ruido No gaussiano
    //***************************************************************************
    if (arg == 1)
    {
float eta[N];

for (i = 0; i < N; i++)
{
    // Generamos ruido no gaussiano y sumamos a RV
    eta[i] = sqrt(noise) * (RandomUniform() - 0.5) * 2;
    rv[i] = rv[i] + eta[i];

    // Escribimos en archivo
    fprintf (ofile, "%f\t%f\t0.0 \n", t[i], rv[i]);
//fprintf (ofile, "%f\t%f\t0.0 \n", t[i], eta[i]); // Solo ruido no gaussiano
}
    }
    //***************************************************************************
    // Serie (sinusoide o RV) más Ruido Blanco Gaussiano
    //***************************************************************************
    if (arg == 2)
    {
double wn[N], Ro, rn_final[N];

// Generamos ruido blanco, sumamos a RV y escribimos en archivo
for (i = 0; i < N; i++)
{
    wn[i] = sqrt(noise) * RandomGaussian (0.0, 1.0);
    rv[i] = rv[i] + wn[i];

    fprintf (ofile, "%lf\t%lf\t0.0 \n", t[i], rv[i]);
    //fprintf (ofile, "%lf\t%lf\t0.0 \n", t[i], wn[i]); // Solo ruido blanco gaussiano
}
    }
    //***************************************************************************
    // Serie (sinusoide o RV) más Ruido Rojo. Proceso AR(1)
```

```
    //*************************************************************************
    if (arg == 3)
    {
double rn[N], wn[N], Rho[N], rn_final[N];

// Generación del ruido rojo, iteración 1
rn[0] = RandomGaussian (0.0, 1.0) * sqrt(1 - exp(-2 * (t[1] - t[0]) / tau));
wn[0] = rn[0];
rn_final[0] = sqrt(noise) * rn[0];
// iteración 2 en adelante
for (k = 1; k < N; k++)
{
    // Ruido blanco con varianza 1 - exp(-2 * (t[k] - t[k-1]) / Tau
    wn[k] = RandomGaussian (0.0, 1.0) * sqrt(1 - exp(-2 * (t[k] - t[k-1]) / tau));
    Rho[k] = exp(-(t[k] - t[k-1]) / tau);
    rn[k] = (Rho[k] * rn[k - 1]) + wn[k];
     rn_final[k] = sqrt(noise) * rn[k];
}

// Sumamos ruido a RV y escribimos en archivo
for (i = 0; i < N; i++)
{
    rv[i] = rv[i] + rn_final[i];
    fprintf (ofile, "%lf\t%lf\t0.0 \n", t[i], rv[i]);
    //fprintf (ofile, "%lf\t%lf\t%lf \n", t[i], rn_final[i], Rho[i]); // Ruido rojo Rn_final[i] y Rho[i]
}
    }
    fclose (ofile);
    return 0;
}
```

## powerspectrum.c

```
#include "rn.h"

/*****************************************************************************
 * FUNCIÓN: Powerspectrum
 *****************************************************************************
 * Función con el LOMB_SCARGLE para calcular el espectro de potencias de una RV,
 * o en general de una serie temporal no equiespaciada en tiempo
 *****************************************************************************
 * Argumentos:
 * file archivo que contiene la serie temporal con la RV
 * arg argumento switch para el flujo de la función, permite guardar en
 *  dos archivos diferentes dependiendo si es el espectro de una RV o
 *  de un ruido
 *****************************************************************************
 * Devuelve el periodo máximo en el espectro. También escribe en archivo el
 * espectro
 *****************************************************************************/
float Powerspectrum (char file[], int arg)
{
    int i, j ,k;
    //*************************************************************************
    // Lectura del archivo con la RV
    //*************************************************************************
    serieTemporal serie = DataRead(file);
    int N = serie.size;

    if (serie.t[0] < 100000 )
    {
        for (i = 0; i < N; i++)
        {
            serie.t[i] = serie.t[i] + 2400000;
        }
    }

// DEBUG PRINT
    //for (i = 0; i < n; i++)
//fprintf (stdout,"serie.t[0] = %lf\t serie.rv[0] = %lf\n", serie.t[0], serie.rv[0]);

    //*************************************************************************
    // LOMB-SCARGLE
    //*************************************************************************
double mean_rv = 0.0;
double mean_t = 0.0;
double q = 0.0;
double tmin, tmax, Pmax, rv2[N], t2[N], LSP, LIP, dP;
int M;
    int Nind = 10000;
double c0, p[Nind], f[Nind], a, b[Nind], c, s, z[Nind], pmax, fmax;
double zmax = 0.0;
```

```
// Busqueda del max y min en t
tmax = tmin = serie.t[0];
for (j = 0; j < N; j++)
{
if (serie.t[j] > tmax) tmax = serie.t[j];
if (serie.t[j] < tmin) tmin = serie.t[j];
}

// Periodo maximo de ensayo
Pmax = 0.5 * (tmax - tmin);

// Medias
for (i = 0; i < N; i++)
{
mean_rv += serie.rv[i];
mean_t += serie.t[i];
}
mean_rv /= N;
mean_t /= N;

for (i = 0; i < N; i++)
{
rv2[i] = serie.rv[i] - mean_rv;
t2[i] = serie.t[i] - mean_t;
  q = q + pow(rv2[i],2);
}

// DEBUG PRINT
//fprintf (stdout,"rv2[0] = %lf\t tmax = %lf\t tmin = %lf\n", rv2[0], tmax, tmin);
//FILE *ofile2;
//ofile2 = fopen("power2.dat", "w");
//for (i = 0; i < n; i++) fprintf (ofile2, "%lf\n", rv2[i]);

LSP = Pmax;                                              // Límite superior del período [d]
LIP = 1.0;                                              // Límite inferior del período [d]
dP = 1.0;                                              // Intervalo del período [d], tamaño de la malla

M = ceil((LSP - LIP)/dP) + 1;                                                  // Número de puntos del spectro
c0 = (N - 1.0)/(N + q);

// DEBUG PRINT
//fprintf (stdout,"n = %d\t\t nCol= %d\t m = %d\t c0 = %lf\n", n, nCol, m, c0);
//fprintf (stdout,"q = %lf\t mean_t = %lf\t mean_rv = %lf\t LSP = %lf\n", q, mean_t, mean_rv, LSP);

for (i = 0; i < M; i++)
{
p[i] = LSP - (i - 1) * dP;
f[i] = 1.0 / p[i];
b[i] = TWOPI * f[i];
c = 0.0;
s = 0.0;
for (j = 0; j < N ; j++)
{
a = b[i] * t2[j];
// DEBUG PRINT
//fprintf (ofile2,"b[%d] = %1.14lf\t t2[%d] = %4.12lf\t a = %lf\n", i, b[i], j, t2[j], a);
s = s + rv2[j] * sin(a);
c = c + rv2[j] * cos(a);
}
// DEBUG PRINT
//fprintf (ofile2,"c = %lf\t s = %lf\n", c, s);

z[i] = (c*c + s*s)*c0;

if (z[i] > zmax)
{
zmax = z[i];
fmax = f[i];
pmax = 1.0/fmax;
}
    }

    //*************************************************************************
    // Creación de archivos data_power.dat o data_power_noisig.dat
    //*************************************************************************
    if (arg == 0)
    {
// No escribir archivo!
}
    if (arg == 1)
    {
        // Salvado de archivo "data_power.dat"
        FILE *ofile;
```

```
        ofile = fopen("data_power.dat", "w");
        for (i = 0; i < M; i++)  fprintf (ofile,"%lf\t%lf\n", f[i], z[i]);
        fclose (ofile);
    }
    if (arg == 2)
    {
        // Salvado de archivo "data_power_noisig.dat"
        FILE *ofile;
        ofile = fopen("data_power_noisig.dat", "w");
        for (i = 0; i < M; i++)  fprintf (ofile,"%lf\t%lf\n", f[i], z[i]);
        fclose (ofile);
    }
fprintf (stdout,"-------------------------------------------------\n");
//fprintf (stdout, "P_lomb1 = %.1lf días      Power = %.1lf\n", pmax, dP, zmax);
fprintf (stdout, "P_lomb1      = %.1lf días \n", pmax, dP);
fprintf (stdout,"-------------------------------------------------\n\n");
return pmax;
}


/*************************************************************************
 * FUNCIÓN: Powerspectrum_lomb
 *************************************************************************
 * Función que llama a la función de MATLAB LOMB basada en numerical recipes
 *************************************************************************
 * Argumentos:
 * arg argumento switch para el flujo de la función, permite imprimir
 *  archivos diferentes si es una serie RV o del ruido extraido
 *************************************************************************
 * Devuelve 0 si finaliza correctamente
 ************************************************************************/
int Powerspectrum_lomb (int arg)
{
if (arg == 1) // Para el espectro de una RV
{
system ("octave --silent matlab_espectro_rv.m");
}
if (arg ==2) // Para el espectro de un ruido
{
system ("octave --silent matlab_espectro_rn.m");
}
if (arg == 3)
{
system ("octave --silent matlab_espectro_rvmod.m");
}

return 0;
}
```

## print_gnuplot.c

```
#include "rn.h"

/*************************************************************************
 * FUNCIÓN: PrintGnuplot
 *************************************************************************
 * Función para la salida gráfica através de GNUPLOT. Utiliza pipes para enviar
 * comando por comando. Genera un multiplot, con la curva RV, la curva en fase y
 * superpuesta la modelada con los parámetros del EXOFIT, el digrama en fase del
 * ruido O-C (RV - RV del ajuste) y por último los powerspectra de la RV y del
 * ruido. La salida gráfica la puede hacer bien por pantalla o a un archivo png
 *************************************************************************
 * Argumentos:
 * arg argumento switch que sirve para controlar el flujo de la función
 * period periodo de la primera RV (necesario para el diagrama en fase)
 * period2 periodo de la segunda RV
 * file nombre del archivo que contiene la primera RV
 * file2 nombre del arhcivo que contiene la segunda RV
 *************************************************************************
 * Devuelve 0 si finaliza correctamente
 ************************************************************************/
int PrintGnuplot (int arg, float period, float period2, char file[], char file2[])
{
    int i;
    FILE *gnuplotPipe = popen ("gnuplot -persistent", "w");

    //************************************************************************
    // Lectura de archivos
    //************************************************************************
    serieTemporal serie = DataRead(file);
    int n = serie.size;
    serieTemporal serie2 = DataRead(file2);
```

```
    //*********************************************************************
    // Diagrama en fase serie1
    //*********************************************************************
    double f[n], f_i[n];
    for (i = 0; i < n; i++)
    {
f_i[i] = floor((serie.t[i] - serie.t[0]) / period); // parte entera
f[i] = ((serie.t[i] - serie.t[0]) - (f_i[i] * period)) / period; // f_i = rem(ti-t0,P)/P
    }
    //*********************************************************************
    // Diagrama en fase serie2
    //*********************************************************************
    double f2[n], f_i2[n];
    for (i = 0; i < n; i++)
    {
f_i2[i] = floor((serie2.t[i] - serie2.t[0]) / period2); // parte entera
f2[i] = ((serie2.t[i] - serie2.t[0]) - (f_i2[i] * period2)) / period2; // f_i = rem(ti-t0,P)/P
    }

    //*********************************************************************
    // Salida por pantalla
    //*********************************************************************
    if (arg == 1 || arg == 3)
    {
fputs ("set terminal wxt 1 size 500,700 title 'RedNoise' \n", gnuplotPipe);
fputs ("set multiplot \n", gnuplotPipe);
fputs ("set size 1,0.33 \n", gnuplotPipe);
fputs ("set rmargin at screen 0.95 \n", gnuplotPipe);
//fputs ("set lmargin 8 \n", gnuplotPipe);
fputs ("set origin 0,0.66 \n", gnuplotPipe);
fputs ("set ylabel 'RV [m/s]' \n", gnuplotPipe);
//fputs ("set yrange [-90:90] \n", gnuplotPipe);
fputs ("set xlabel 'Date [JD]' \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("set format x '%7.0f' \n", gnuplotPipe);
fputs ("set xtics font 'Times-Roman, 8' \n", gnuplotPipe);
fputs ("set mxtics 2 \n", gnuplotPipe);
fputs ("plot '-' using 1:2 w p pointtype 7 pointsize 0.4 lt 3 \n", gnuplotPipe);
for (i = 0; i < n; i++)  fprintf (gnuplotPipe,"%lf %lf\n", serie.t[i], serie.rv[i]);
fputs ("e", gnuplotPipe);
    }
    //*********************************************************************
    // Salida a archivo png
    //*********************************************************************
    if (arg == 2 || arg == 4)
    {
fputs ("set terminal pngcairo size 650,950 \n", gnuplotPipe);
fputs ("set output 'plot1.png' \n", gnuplotPipe);
fputs ("set multiplot \n", gnuplotPipe);
fputs ("set size 1,0.33 \n", gnuplotPipe);
fputs ("set origin 0,0.66 \n", gnuplotPipe);
fputs ("set ylabel 'RV [m/s]' \n", gnuplotPipe);
//fputs ("set yrange [-90:90] \n", gnuplotPipe);
fputs ("set xlabel 'Date [JD]' \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("set format x '%7.0f' \n", gnuplotPipe);
fputs ("set xtics font 'Times-Roman, 8' \n", gnuplotPipe);
fputs ("set xtics font 'Times-Roman, 10' \n", gnuplotPipe);
fputs ("set mxtics 2 \n", gnuplotPipe);
fputs ("plot '-' using 1:2 w p pointtype 7 pointsize 0.4 lt 3 \n", gnuplotPipe);
for (i = 0; i < n; i++)  fprintf (gnuplotPipe,"%lf\t%lf\n", serie.t[i], serie.rv[i]);
fputs ("e", gnuplotPipe);
    }
    if (arg == 1 || arg == 2)
    {
//*********************************************************************
// Diagrama en fase
//*********************************************************************
fputs ("reset \n", gnuplotPipe);
fputs ("set size 1,0.33 \n", gnuplotPipe);
fputs ("set origin 0,0.33 \n", gnuplotPipe);
fputs ("set ylabel 'RV [m/s]' \n", gnuplotPipe);
fputs ("set yrange [*:*] \n", gnuplotPipe);
fputs ("set xlabel 'Phase' \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("set xtics font 'Times-Roman, 10' \n", gnuplotPipe);
fputs ("set format x '%g' \n", gnuplotPipe);
fputs ("set mxtics 1 \n", gnuplotPipe);
fputs ("plot '-' using 1:2 w p pointtype 7 pointsize 0.4 lt 3 \n", gnuplotPipe);
for (i = 0; i < n; i++)  fprintf (gnuplotPipe,"%lf\t%lf\n", f[i], serie.rv[i]);
fputs ("e", gnuplotPipe);
//*********************************************************************
// Powerspectrum
```

```
//*********************************************************************
fputs ("reset \n", gnuplotPipe);
fputs ("set size 1,0.33 \n", gnuplotPipe);
fputs ("set origin 0,0 \n", gnuplotPipe);
fputs ("set lmargin 9 \n", gnuplotPipe);
fputs ("set xlabel 'Frequency [1/d]' \n", gnuplotPipe);
fputs ("set ylabel 'Power [m/s^2]' \n", gnuplotPipe);
fputs ("set ylabel offset -1 \n", gnuplotPipe);
fputs ("set yrange [*:*] \n", gnuplotPipe);
//fputs ("set logscale x \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("plot 'data_power.dat' using 1:2 w linespoints pointtype 7 pointsize 0.4 lt 3 \n", gnuplotPipe);
fputs ("unset multiplot \n", gnuplotPipe);
    }
    if (arg == 3 || arg == 4)
    {
//*********************************************************************
// Diagrama en fase
//*********************************************************************
fputs ("reset \n", gnuplotPipe);
fputs ("set size 1,0.22 \n", gnuplotPipe);
fputs ("set origin 0,0.44 \n", gnuplotPipe);
fputs ("set ylabel 'RV [m/s]' \n", gnuplotPipe);
fputs ("unset xlabel \n", gnuplotPipe);
fputs ("set yrange [*:*] \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("unset xtics \n", gnuplotPipe);
fputs ("plot '-' w p pointtype 7 pointsize 0.4 lt 3\n", gnuplotPipe);
for (i = 0; i < n; i++)
{
    fprintf (gnuplotPipe,"%lf\t%lf\n", f[i], serie.rv[i]);
    fprintf (gnuplotPipe,"%lf\t%lf\n", f2[i], serie2.other[i]);
}
fputs ("e", gnuplotPipe);
//*********************************************************************
// Diagrama O-C
//*********************************************************************
fputs ("reset \n", gnuplotPipe);
fputs ("set size 1,0.15 \n", gnuplotPipe);
fputs ("set lmargin 9.4 \n", gnuplotPipe);
fputs ("set origin 0,0.33 \n", gnuplotPipe);
fputs ("set ylabel 'O-C' \n", gnuplotPipe);
fputs ("set ylabel offset '-4' \n", gnuplotPipe);
fputs ("set yrange [*:*] \n", gnuplotPipe);
fputs ("set xlabel 'Phase' \n", gnuplotPipe);
fputs ("set xzeroaxis linewidth 2.0 \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("set xtics font 'Times-Roman, 10' \n", gnuplotPipe);
fputs ("set format x '%g' \n", gnuplotPipe);
fputs ("set mxtics 1 \n", gnuplotPipe);
fputs ("set ytics format ' ' \n", gnuplotPipe);
fputs ("plot '-' using 1:2 w p pointtype 7 pointsize 0.3\n", gnuplotPipe);
for (i = 0; i < n; i++)  fprintf (gnuplotPipe,"%lf\t%lf\n", f2[i], serie2.rv[i]);
fputs ("e", gnuplotPipe);
//*********************************************************************
// Powerspectrum de la RV con el powerspectrum del ruido
//*********************************************************************
fputs ("reset \n", gnuplotPipe);
fputs ("set size 1,0.33 \n", gnuplotPipe);
fputs ("set origin 0,0 \n", gnuplotPipe);
fputs ("unset xzeroaxis \n", gnuplotPipe);
fputs ("set xlabel 'Frequency [1/d]' \n", gnuplotPipe);
//fputs ("set xrange [-0.005:0.2] \n", gnuplotPipe);
fputs ("set yrange [*:*] \n", gnuplotPipe);
fputs ("set ylabel 'Power [m/s^2]' \n", gnuplotPipe);
fputs ("set ylabel offset -1 \n", gnuplotPipe);
fputs ("set format y '%g' \n", gnuplotPipe);
//fputs ("set logscale y \n", gnuplotPipe);
//fputs ("set logscale x \n", gnuplotPipe);
fputs ("unset key \n", gnuplotPipe);
fputs ("plot 'data_power.dat' using 1:2 w linespoints pointtype 7 pointsize 0.4 linecolor 3, 'data_power_noisig_fnyq.dat' using 1:2 w linespoints poi
//fputs ("plot 'data_power_noisig.dat' using 1:2 w linespoints pointsize 0.3 linewidth 0.5 linecolor 1 \n", gnuplotPipe);
fputs ("unset multiplot \n", gnuplotPipe);
    }
    pclose (gnuplotPipe);
    return 0;
}


/*************************************************************************
 * FUNCIÓN: PrintGnuplotRn
 *************************************************************************
 * Función para la salida gráfica a través de GNUPLOT. Genera un plot con el
```

```
 * espectro de la RV, con el espectro teórico del ruido (rojo) y con el límite
 * para el test de significancia de los picos de la RV. La salida la hace bien
 * sea por pantalla o a un arhcivo png
 ***************************************************************************
 * Argumentos:
 * arg argumento switch para controlar el flujo de la función
 * alpha el parámetro de incertidumbre
 ***************************************************************************
 * Devuelve 0 si finaliza correctamente
 ***************************************************************************/
int PrintGnuplotRn (int arg, float alpha)
{
    FILE *gnuplotPipe = popen ("gnuplot -persistent", "w");
    if (arg == 1)  fprintf (gnuplotPipe,"set terminal wxt 1 size 900,600 title 'RedNoise' \n ");
    if (arg == 2)
    {
fprintf (gnuplotPipe,"set terminal pngcairo size 900,600 \n");
fprintf (gnuplotPipe,"set output 'plot3.png' \n");
    }
    fprintf (gnuplotPipe,"set xlabel 'Frequency [1/d]' \n ");
    //fprintf (gnuplotPipe,"set xrange [-0.005:0.1] \n ");
    fprintf (gnuplotPipe,"set ylabel 'Power [m/s^2]' \n ");
    fprintf (gnuplotPipe,"set ylabel offset 1 \n");
    fprintf (gnuplotPipe,"plot 'data_power.dat' u 1:2 title 'RV Spectrum' w lp pt 7 ps 0.5 lc 3, 'data_power_noisig_test.dat' u 1:2 title 'Null Red Noise Spectrum
    pclose (gnuplotPipe);
    return 0;
}
```

## randomlib.c

```
#include "rn.h"

/***************************************************************************
 * FUNCIÕN: varias. Libería
 ***************************************************************************
 * Librería para la generación de números pseudo-aleatórios, también tiene una
 * implementación de la FFT
 ***************************************************************************
 ***************************************************************************/
#define FALSE 0
#define TRUE 1

/*
   This Random Number Generator is based on the algorithm in a FORTRAN version
   published by George Marsaglia and Arif Zaman, Florida State University; ref.:
   see original comments below. At the fhw (Fachhochschule Wiesbaden, W.Germany)
   Dept. of Computer Science, we have written sources in further languages
   (C, Modula-2 Turbo-Pascal(3.0, 5.0), Basic and Ada) to get exactly the same
   test results compared with the original FORTRAN version. April 1989
   Karl-L. Noell <NOELL@DWIFH1.BITNET> and  Helmut  Weber <WEBER@DWIFH1.BITNET>

   This random number generator originally appeared in "Toward a Universal
   Random Number Generator" by George Marsaglia and Arif Zaman. Florida State
   University Report: FSU-SCRI-87-50 (1987) It was later modified by F. James
   and published in "A Review of Pseudo-random Number Generators"
   THIS IS THE BEST KNOWN RANDOM NUMBER GENERATOR AVAILABLE.
   (However, a newly discovered technique can yield a period of 10^600. But that
   is still in the development stage.)
   It passes ALL of the tests for random number generators and has a period of
   2^144, is completely portable (gives bit identical results on all machines
   with at least 24-bit mantissas in the floating point representation).
   The algorithm is a combination of a Fibonacci sequence (with lags of 97 and
   33, and operation "subtraction plus one, modulo one") and an "arithmetic
   sequence" (using subtraction).

   Use IJ = 1802 & KL = 9373 to test the random number generator. The subroutine
   RANMAR should be used to generate 20000 random numbers. Then display the next
   six random numbers generated multiplied by 4096*4096 If the random number
   generator is working properly, the random numbers should be:
   6533892.0  14220222.0  7275067.0
   6172232.0  8354498.0   10633180.0
*/

/* Globals */
double u[97],c,cd,cm;
int i97,j97;
int test = FALSE;

/*
   This is the initialization routine for the random number generator. NOTE: The
   seed variables can have values between:
   0 <= IJ <= 31328        0 <= KL <= 30081
   The random number sequences created by these two seeds are of sufficient
```

```
   length to complete an entire calculation with.
   For example, if sveral different groups are working on different parts of the
   same calculation, each group could be assigned its own IJ seed. This would
   leave each group with 30000 choices for the second seed. That is to say, this
   random number generator can create 900 million different subsequences -- with
   each subsequence having a length of approximately 10^30.
*/
void RandomInitialise(int ij,int kl)
{
   double s,t;
   int ii,i,j,k,l,jj,m;

   /*
      Handle the seed range errors. First random number seed must be between 0
      and 31328
      Second seed must have a value between 0 and 30081
   */
   if (ij < 0 || ij > 31328 || kl < 0 || kl > 30081) {
ij = 1802;
kl = 9373;
   }

   i = (ij / 177) % 177 + 2;
   j = (ij % 177)       + 2;
   k = (kl / 169) % 178 + 1;
   l = (kl % 169);

   for (ii=0; ii<97; ii++) {
      s = 0.0;
      t = 0.5;
      for (jj=0; jj<24; jj++) {
         m = (((i * j) % 179) * k) % 179;
         i = j;
         j = k;
         k = m;
         l = (53 * l + 1) % 169;
         if (((l * m % 64)) >= 32)
            s += t;
         t *= 0.5;
      }
      u[ii] = s;
   }

   c    = 362436.0 / 16777216.0;
   cd   = 7654321.0 / 16777216.0;
   cm   = 16777213.0 / 16777216.0;
   i97  = 97;
   j97  = 33;
   test = TRUE;
}

/*
   This is the random number generator proposed by George Marsaglia in Florida
   State University Report: FSU-SCRI-87-50
*/
double RandomUniform(void)
{
   double uni;

   /* Make sure the initialisation routine has been called */
   if (!test)
    RandomInitialise(1802,9373);

   uni = u[i97-1] - u[j97-1];
   if (uni <= 0.0)
      uni++;
   u[i97-1] = uni;
   i97--;
   if (i97 == 0)
      i97 = 97;
   j97--;
   if (j97 == 0)
      j97 = 97;
   c -= cd;
   if (c < 0.0)
      c += cm;
   uni -= c;
   if (uni < 0.0)
      uni++;

   return(uni);
}
```

```
/*
  ALGORITHM 712, COLLECTED ALGORITHMS FROM ACM. THIS WORK PUBLISHED IN
  TRANSACTIONS ON MATHEMATICAL SOFTWARE, VOL. 18, NO. 4, DECEMBER, 1992,
  PP. 434-435.
  The function returns a normally distributed pseudo-random number with a given
  mean and standard deviation.  Calls are made to a function subprogram which
  must return independent random numbers uniform in the interval (0,1). The
  algorithm uses the ratio of uniforms method of A.J. Kinderman and J.F. Monahan
  augmented with quadratic bounding curves.
*/
double RandomGaussian(double mean,double stddev)
{
   double  q,u,v,x,y;

   /*
   Generate P = (u,v) uniform in rect. enclosing acceptance region
   Make sure that any random numbers <= 0 are rejected, since gaussian()
   requires uniforms > 0, but RandomUniform() delivers >= 0.
   */
   do {
      u = (RandomUniform()-0.5) * 2;
      v = (RandomUniform()-0.5) * 2;
    if (u <= 0.0 || v <= 0.0)
    {
        u = 1.0;
        v = 1.0;
    }
      v = 1.7156 * (v - 0.5);

      /*  Evaluate the quadratic form */
      x = u - 0.449871;
      y = fabs(v) + 0.386595;
      q = x * x + y * (0.19600 * y - 0.25472 * x);

      /* Accept P if inside inner ellipse */
      if (q < 0.27597)
break;

      /*  Reject P if outside outer ellipse, or outside acceptance region */
    } while ((q > 0.27846) || (v * v > -4.0 * log(u) * u * u));

    /*  Return ratio of P's coordinates as the normal deviate */
    return (mean + stddev * v / u);
}

/*
   Return random integer within a range, lower -> upper INCLUSIVE
*/
int RandomInt(int lower,int upper)
{
   return((int)(RandomUniform() * (upper - lower + 1)) + lower);
}

/*
   Return random float within a range, lower -> upper
*/
double RandomDouble(double lower,double upper)
{
   return((upper - lower) * RandomUniform() + lower);
}

/*---------------------------------------------------------------------------
   This computes an in-place complex-to-complex FFT. x and y are the real and
   imaginary arrays of 2^m points.
   dir =  1 gives forward transform, dir = -1 gives reverse transform

     Formula: forward
                  N-1
                  ---
              1   \          - j k 2 pi n / N
      X(n) = ---   >   x(k) e                    = forward transform
              N   /                                n=0..N-1
                  ---
                  k=0

      Formula: reverse
                  N-1
                  ---
                  \          j k 2 pi n / N
      X(n) =       >   x(k) e                    = forward transform
                  /                                n=0..N-1
                  ---
                  k=0
```

```
*/
int FFT(int dir,int m,double *x,double *y)
{
   long nn,i,i1,j,k,i2,l,l1,l2;
   double c1,c2,tx,ty,t1,t2,u1,u2,z;

   /* Calculate the number of points
   nn = 1;
   for (i=0;i<m;i++)
      nn *= 2;
*/
nn = 1 << m;

   /* Do the bit reversal */
   i2 = nn >> 1;
   j = 0;
   for (i=0;i<nn-1;i++) {
      if (i < j) {
         tx = x[i];
         ty = y[i];
         x[i] = x[j];
         y[i] = y[j];
         x[j] = tx;
         y[j] = ty;
      }
      k = i2;
      while (k <= j) {
         j -= k;
         k >>= 1;
      }
      j += k;
   }

   /* Compute the FFT */
   c1 = -1.0;
   c2 = 0.0;
   l2 = 1;
   for (l=0;l<m;l++) {
      l1 = l2;
      l2 <<= 1;
      u1 = 1.0;
      u2 = 0.0;
      for (j=0;j<l1;j++) {
         for (i=j;i<nn;i+=l2) {
            i1 = i + l1;
            t1 = u1 * x[i1] - u2 * y[i1];
            t2 = u1 * y[i1] + u2 * x[i1];
            x[i1] = x[i] - t1;
            y[i1] = y[i] - t2;
            x[i] += t1;
            y[i] += t2;
 }
 z =  u1 * c1 - u2 * c2;
         u2 = u1 * c2 + u2 * c1;
         u1 = z;
      }
      c2 = sqrt((1.0 - c1) / 2.0);
      if (dir == 1)
         c2 = -c2;
      c1 = sqrt((1.0 + c1) / 2.0);
   }

   /* Scaling for forward transform */
   if (dir == 1) {
      for (i=0;i<nn;i++) {
         x[i] /= (double)nn;
         y[i] /= (double)nn;
      }
   }

   return(TRUE);
}
```

## README.txt

```
 ------------------------ ACERCA DE REDNOISE --------------------------------

 RedNoise es programa desarrollado Carlos A. Gómez bajo la supervisión de
 Luis Dinis y Jose Caballero como Trabajo de Fin de máster (máster inter-
 universitario en Astrofísica UAM/UCM Madrid 2012-2013).

 Este programa permite el modelado curvas de velocidad radial de una estrella
```

```
con un planeta, con la posibilidad de añadir distintos tipos de ruido. Con una
implementación del algoritmo LOMB-SCARGLE el programa puede encontrar la
frecuencia con mayor significación estadística en esta señal (período orbital
del planeta si se trata de una curva RV). Se le presta central atención a la
generación del ruido rojo como un proceso autorregresivo de primer orden AR(1)
y a su posterior extracción de la señal. Los parámetros del espectro de este
ruido rojo extraido pueden ser ajustados para calcular un espectro teórico y
unos límites de significación con que comparar los picos de la señal RV
modelada.

La guía de compilación del programa se encuentra más abajo. RedNoise puede ser
invocado en modo consola o ejecutado en modo GUI que facilita la introducción
de los parámetros.

El programa está profusamente comentado y en su desarrollo se ha aplicado un
rudimentario pero efectivo sistema de control de versiones (manual).
RedNoise está desarrollado en el lenguaje C en un entorno Linux e incluye unas
rutinas en MATLAB (para el ajuste de los parámetros del espectro del ruido
rojo). También se hace uso de EXOFIT (www.star.ucl.ac.uk/~lahav/exofit.html?)
para el ajuste de los parámetros orbitales de las curvas RV.

La ayuda del programa en modo consola se encuentra más abajo con una
descripción de los parámetros y las opciones del programa. Para ejecutar el
GUI basta con ejecutar: ./rednoise en el shell desde la carpeta raíz del
programa, o dar doble click sobre el archivo. El GUI está escrito en YAD que
corre en cualquier distro UNIX-LINUX, para su funcionamiento este (YAD) debe
estar instalado. La salida de gráficos se hace através de GNUPLOT que va
preinstalado en la mayoría de sistemas UNIX-LINUX.

------------------------- GUÍA DE COMPILACIÓN --------------------------------

En el directorio raíz de la aplicación ejecutar (en el shell) el comando:
./make && make clean

---------------------- GUÍA DE USO EN MODO CONSOLA --------------------------

Parámetros (argumentos en modo consola):

t_0 :       tiempo inicial JD [d], de tipo (double)
t_f :       tiempo final JD [d], de tipo (double)
P :         periodo orbital [d], de tipo (float)
K_1 :       semiamplitud de la RV [m/s], de tipo (float)
N :         numero de puntos, de tipo (int)
gamma :     RV_0, velocidad radial del sistema (centro de masas) [km/s],
            de tipo (float)
phi :       desfase [grados], de tipo (float)
noise :     amplitud del ruido, de tipo (float)
e :         excentricidad de la órbita, de tipo (float)
m2sinI :    masa del planeta por el seno del ángulo de inclinación [m jup],
            de tipo (float)
m1 :        masa de la estrella [m sol], de tipo (float)
w :         argumento del periastro [grados], de tipo (float)
tau :       parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
ventana :   ventana de suavizado para el algoritmo CMA

Rho :       parámetro del ruido rojo. está relacionado con su memoria, depende
            de tau
A :         parámetro del ruido rojo. factor relacionado con su amplitud,
            depende de noise
fNyq :      frecuencia de Nyquist (sampleado) en la señal de la RV modelada
alpha :     incertidumbre para el test de confiabilidad
-------------------------------------------------------------------------------
Para procesar una RV guardada en archivo de texto plano:
./rn --file
Ejemplo: ./rn --file data/hd_8574.dat

En el directorio data hay una compilación de curvas RV reales tomadas de la
literatura disponible.
-------------------------------------------------------------------------------
Para generar una sinusoide con ruido no gaussiano:
./rn --sin 0 t_0 t_f P N K_1 gamma phi noise
Ejemplo: ./rn --sin 1 53280 53334 15.0 50 58.3 0.0 0.0 5.0

Para generar una sinusoide con ruido blanco gaussiano:
./rn --sin 1 t_0 t_f P N K_1 gamma phi noise
Ejemplo: ./rn --sin 2 53280 53334 15.0 50 58.3 0.0 0.0 5.0

Para generar una sinusoide con ruido rojo:
./rn --sin 2 t_0 t_f P N K_1 gamma phi noise tau
Ejemplo: ./rn --sin 3 53280 53334 15.0 50 58.3 0.0 0.0 5.0 1.0
-------------------------------------------------------------------------------
Para generar una curva RV con ruido no gaussiano:
./rn --rv 0 t_0 t_f P N gamma e m2sinI m1 w noise
```

```
  Ejemplo: ./rn --rv 1 53280 53334 15.0 50 0.0 0.4 0.9 0.9 90.0 5.0

  Para generar una curva RV con ruido blanco gaussiano:
  ./rn --rv 1 t_0 t_f P N gamma e m2sinI m1 w noise
  Ejemplo: ./rn --rv 2 53280 53334 15.0 50 0.0 0.4 0.9 0.9 90.0 5.0

  Para generar una curva RV con ruido rojo:
  ./rn --rv 2 t_0 t_f P N gamma e m2sinI m1 w noise tau
  Ejemplo: ./rn --rv 3 53280 53334 15.0 50 0.0 0.4 0.9 0.9 90.0 5.0 1.0
  ------------------------------------------------------------------------------
  Para extraer el ruido rojo, introducir los parámetros orbitales que arroja el
  EXOFIT:
  ./rn --rnext P K_1 gamma e w ventana
  Ejemplo: ./rn --rnext 99.84132 51.73582 1.21484 0.08851 3.297 0

  Para el ajuste de los parámetros del ruido rojo (Es necesario editar los
  archivos FIT.m y FIT_function.m para cambiar ciertos parámetros antes de
  utilizar esta opción):
  ./rn --rnfit

  Para el test de sginificación de los picos de la RV:
  ./rn --rntest Rho A fNyq alpha
  Ejemplo: ./rn --rntest 0.4173 1.278 0.100658 0.01

  ---------------------------- METODOLOGÍA -----------------------------------

  La metodología básica de trabajo para el test de significación es así:

  Generar curva:
  ./rn --rv 2 t_0 t_f P N gamma e m2sinI m1 w noise tau

  Ajustar de parametros orbitales en EXOFIT
  Copiar archivo a data_curve_mod.dat

  Copiar data_curve_mod.dat de vuelta al directorio raíz
  Extraer el ruido:
  ./rn --rnext P K_1 gamma e w ventana

  Cambiar parámetros del FIT en MATLAB
  Ajustar espectro ruido rojo:
  ./rn --rnfit

  Generar el test de significación de los picos de la RV:
  ./rn --rntest Rho A fNyq alpha
  ------------------------------------------------------------------------------
```

## rednoise

```
#!/bin/bash

END=false
FILE=""
TITLE="--title="RedNoiseGUI""
PAR="--on-top --window-icon="rednoiseicon.png" --button="gtk-ok:0" "

while [ "$END" = "false" ]
do
    gnup=`ps cax | grep gnuplot | grep -vc grep`
    if [ $gnup != 0 ]; then
        killall gnuplot
    fi

    MENU=`yad --list $TITLE $PAR --text=" " --height="240" --width="300" --separator=" " --column="Opciones" "Procesar Archivo RV con el Lomb-Scargle
    if [ "$?" != 0 ]; then
        END=true
    fi

    case $MENU in
        "Procesar Archivo RV con el Lomb-Scargle ")
            FILE=`yad --file $TITLE $PAR --height="520" --width="600" --text="Seleccionar archivo con curva RV" --filename="/home/carlos/Escritorio/V
            ./rn --file $FILE | yad --title="${FILE:54:65}" $PAR --fontname="Monospace 9" --text-info --height="170" --width="360"
            ;;

        "Generar Sinusoide con Ruido y procesar con LS ")
            FORM=`yad --form $TITLE $PAR --text="---- Parámetros de la sinusoide:" --field=" Fecha Inicial [JD]" --field=" Fecha Final [JD]" --field=
if [ $? == 0 ]; then
t_0=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $1 }')
t_f=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $2 }')
N=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $3 }')
P=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $4 }')
K_1=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $5 }')
gamma=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $6 }')
phi=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $7 }')
```

```
noise=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $10 }')
tau=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $11 }')
tiporuido=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $12 }')
fi
if [ "$tiporuido" == "Ruido no Gaussiano" ]; then
tiporuido="1"
fi
if [ "$tiporuido" == "Ruido Blanco Gaussiano" ]; then
tiporuido="2"
fi
if [ "$tiporuido" == "Ruido Rojo" ]; then
tiporuido="3"
fi

./rn --sin $tiporuido $t_0 $t_f $P $N $K_1 $gamma $phi $noise $tau | yad --title="Sinusoide generada" $PAR --text-info --fontname="Monospace 9" --height="330" --w
            ;;

        "Generar Curva RV con Ruido y procesar con LS ")
FORM=`yad --form $TITLE $PAR --text="---- Parámetros de la RV:" --field=" Fecha Inicial [JD]" --field=" Fecha Final [JD]" --field=" Número de puntos" --field=" Pe
if [ $? == 0 ]; then
t_0=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $1 }')
t_f=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $2 }')
N=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $3 }')
P=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $4 }')
gamma=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $5 }')
e=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $8 }')
m2sinI=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $9 }')
m1=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $10 }')
w=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $11 }')
noise=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $14 }')
tau=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $15 }')
tiporuido=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $16 }')
fi
if [ "$tiporuido" == "Ruido no Gaussiano" ]; then
tiporuido="1"
fi
if [ "$tiporuido" == "Ruido Blanco Gaussiano" ]; then
tiporuido="2"
fi
if [ "$tiporuido" == "Ruido Rojo" ]; then
tiporuido="3"
fi

./rn --rv $tiporuido $t_0 $t_f $P $N $gamma $e $m2sinI $m1 $w $noise $tau | yad --title="RV generada" $PAR --text-info --fontname="Monospace 9" --height="380" --w
;;

        "Ajuste de los parámetros del Ruido Rojo ")
            FORM=`yad --form $TITLE $PAR --text="---- Parámetros del ajuste de la RV:" --field=" Período (P, T) [d]" --field=" Semiamplitud (K_1) [m/s]" --field="
if [ $? == 0 ]; then
P=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $1 }')
K_1=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $2 }')
gamma=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $3 }')
e=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $4 }')
w=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $5 }')
                ventana=$(echo "$FORM" | awk 'BEGIN { FS = "|" } ; { print $6 }')
            fi

            ./rn --rvfit $P $K_1 $gamma $e $w $ventana | yad $TITLE $PAR --fontname="Monospace 9" --text-info --height="210" --width="360"
            ;;

        "Readme - Ayuda ")
            ./rn --help | yad $TITLE $PAR --text-info --fontname="Monospace 9" --height="600" --width="590"
;;

        "Salir ") END=true
            ;;

    esac
done
exit 0
```

## rn.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>
#include <sys/types.h>
#include <assert.h>

#define VERSION 1.1
```

```
#define TWOPI 6.28318530718
#define PI 3.14159265359
#define G 6.67384e-11        // G en m^3 * kg^-1 * s^?2
#define MSUN_KG 1.98892e30
#define MJUP_KG 1.899e27
#define DAY_SEC 86400.0
#define INDEXMAX 10000

typedef struct {
    int size;
    double t[INDEXMAX];
    double rv[INDEXMAX];
    float other[INDEXMAX];
}serieTemporal;

serieTemporal DataRead(char file[]);
char* ReadFile(char *filename);
int  SinCurve(double t_0,double t_f,float P,int N,float K_1,float gamma,float phi,float noise,float tau,int arg);
int  RvCurve(double t_0,double t_f,float P,int N,float gamma,float e,float m2sinI,float m1,float w,float noise,float tau,int arg);
int RnExtract(float P,float K_1,float gamma,float e,float w,int ventana);
int RvCurveFit(void);
serieTemporal RnTest(float Rho,float A,float fNyq,float alpha);
serieTemporal RnTestMultiple(float Rho,float A,float fNyq,float alpha);
float Powerspectrum(char file[],int arg);
int Powerspectrum_lomb (int arg);
int Noise(int arg,int N,float noise,float tau,double *t,double *rv);
int Cma(char file[],int ventana);
int Cma2(char file[],int ventana);
int PrintGnuplot(int arg,float period,float period2,char file[],char file2[]);
int PrintGnuplotRn(int arg,float alpha);
void GausTest(float mean,float stddev);

// Initialization routine for the random number generator
void RandomInitialise(int ij,int kl);
// Random number generator proposed by George Marsaglia in Florida State University Report: FSU-SCRI-87-50
double RandomUniform(void);
// ALGORITHM 712, normally distributed pseudo-random number generator
double RandomGaussian(double mean,double stddev);
// Random integer within a range
int RandomInt(int lower,int upper);
// Random float within a range
double RandomDouble(double lower,double upper);
// In-place complex-to-complex FFT
int FFT(int dir,int m,double *x,double *y);
```

## rv_curve.c

```
#include "rn.h"

/*****************************************************************************
 * FUNCIÕN: RvCurve
 *****************************************************************************
 * Función para la generación de curvas RV modelando a partir de los parámetros
 * orbitales. Esta función al final llama a la función de generación de ruido
 *****************************************************************************
 * Argumentos:
 * t_0 tiempo inicial de la serie temporal
 * t_f tiempo final de la serie temporal
 * P período orbital del planeta
 * N número de puntos de la curva (medidas RV)
 * gamma RV_0, velocidad sistémica o velocidad radial del centro de masa del
 *  sistema estrella-planeta
 * e excentricidad de la órbita del planeta
 * m2sinI masa del planeta por el seno del ángulo de inclinación de la órbita
 * m1 masa de la estrella
 * w omega pequeña, argumento del periastro
 * noise amplitud del ruido
 * tau parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *  se usa en el caso de querer generar ruido rojo
 * arg argumento switch para escoger el tipo de ruido a generar
 *****************************************************************************
 * Devuelve 0 si finaliza correctamente
 *****************************************************************************/
int RvCurve (double t_0, double t_f, float P, int N, float gamma, float e, float m2sinI, float m1, float w, float noise, float tau, int arg)
{
int i, j, k;
double t[N], M, N_0, E0, E, v, rv[N], K_1, cosv, sinv, dt, swap, ee;
float cosE;
double w_rad = (w * PI) / 180.0;
    float P_sec = P * DAY_SEC;
    float gamma_ms = gamma * 1000.0;
```

```
double t_sec[N], k_factor;
    double m1_kg = m1 * MSUN_KG;
double m2sinI_kg = m2sinI * MJUP_KG;

//**********************************************************************
// Generamos el t aleatoriamente
    //**********************************************************************
// Inicializacion de la semilla y los aleatorios
double seed = time(NULL) % 30000;
RandomInitialise (seed, seed+100);

//~ dt = (t_f - t_0) / N;
for (i = 0; i < N; i++)
{
//t = t_0 + i * dt;
//t[i] = t_0 + RandomUniform() + i * dt;
t[i] = t_0 + RandomUniform() * (t_f - t_0);
}
// Ordenamos el array de t, con algoritmo Bubble sort
for (i = 0; i < N - 1; i++)
{
for (j = 0 ; j < N - i - 1; j++)
{
if (t[j] > t[j+1])  // descendente, usar: <
{
swap = t[j];
t[j] = t[j+1];
t[j+1] = swap;
}
}
}
    //**********************************************************************
// Generación de la RV
    //**********************************************************************
fprintf (stdout,"------------------------------------------------ \n");
// Estimación de el semi-eje mayor despreciando la masa del planeta en comparación a la de la estrella
float a = pow((G * m1_kg * P_sec * P_sec)/(4 * PI * PI), 0.333333);

fprintf (stdout,"a            = %.4f UA \n", a / 149597870700);

    m1_kg = pow (m1_kg, 0.666666);                                          // elevamos la masa de la estrella a 2/3
    // Ecuacion para la media de movimiento (mean motion)
N_0 = TWOPI / P_sec;

k_factor = pow (N_0 * G, 0.333333);                                     // ((2*pi*G)/P)^1/3
    ee = 1 - (e * e);
    t_0 = t_0 * DAY_SEC;

// Ecuacion para la semiamplitud de la RV
K_1 = k_factor * (m2sinI_kg / (m1_kg * (sqrt(ee))));

    fprintf (stdout,"K_1          = %.1f m/s \n", K_1);

    for (i = 0; i < N; i++)
{
        t_sec[i] = t[i] * DAY_SEC;
// Ecuacion para la anomalia media
M = N_0 * (t_sec[i] - t_0);                 // t_0 momento de paso por el perecentro

        // Solucion iterativa de la ecuación de Kepler para la anomalia excentrica
E0 = M;
E = M + e * sin(E0);
while (abs(E - E0) > 0.000001)
{
E0 = E;
E = M + e * sin(E0);
}

        cosE = cos(E);
// Ecuaciones para la anomalia verdadera
cosv = (cosE - e) / (1 - e * cosE);       // cosv=0.5=> v=60,-60
sinv = (sqrt(ee) * sin(E)) / (1 - e * cosE);
if (sinv > 0) v = acos(cosv);
else v = -acos(cosv);

// Ecuacion para la RV
rv[i] = gamma_ms + K_1 * (cos(w_rad + v) + e * cos(w_rad));
}
    //**********************************************************************
// Llamada a la función de generación de ruido
    //**********************************************************************
    if (arg == 1)                                                           // Para ruido no Gaussiano
    {
```

```
        Noise (arg, N, noise, tau, t, rv);
    }
    if (arg == 2)                                                    // Para ruido blanco Gaussiano
    {
        Noise (arg, N, noise, tau, t, rv);
    }
    if (arg == 3)                                                    // Para ruido rojo
    {
        Noise (arg, N, noise, tau, t, rv);
    }

    return 0;
}
```

## rvfit_rnext_rnfit.c

```
#include "rn.h"

/*****************************************************************************
 * FUNCIÓN: RvCurveFit
 *****************************************************************************
 * Función para el ajuste de la RV, prepara el archivo con la serie temporal y
 * da información sobre cómo correr el EXOFIT. Para leer los resultados que
 * arroja el EXOFIT (ejecutar ciertos comandos en R)es necesario leer la ayuda
 * de este, que se encuentra en su respectivo directorio
 *****************************************************************************
 * No recibe argumentos
 *****************************************************************************
 * Devuelve 0 si sale correctamente
 *****************************************************************************/
int RvCurveFit (void)
{
    system ("cp 'data_curve.dat' ./data/fit/data_curve_mod.dat");
    fprintf (stdout,"  Ir a directorio: ~/ExoFit.v2 \n");
    fprintf (stdout,"  Allí ejecutar EXOFIT: ./exofit data/data/fit/data_curve_mod.dat \n");
    fprintf (stdout,"  Esperar 10+ min. \n");
    return 0;
}

/*****************************************************************************
 * FUNCIÓN: RnExtract
 *****************************************************************************
 * Función que extrae el ruido rojo de la curva RV, utilizando los parámetros
 * del ajuste que arroja EXOFIT
 *****************************************************************************
 * Argumentos:
 * P periodo (T1 en la nomenclatura de EXOFIT)
 * K_1 semiamplitud de la RV (K1 en la nomenclatura de EXOFIT)
 * gamma RV_0, velocidad sistémica o velocidad radial del centro de masa del
 *  sistema estrella-planeta (V en la nomenclatura de EXOFIT)
 * e excentricidad de la órbita del planeta (e1 en EXOFIT)
 * w omega pequeña o argumento del periastro (w1 en EXOFIT)
 * ventana ventana de suavizado para el algoritmo CMA
 *****************************************************************************
 * Devuelve 0 si sale correctamente
 *****************************************************************************/
int RnExtract (float P, float K_1, float gamma, float e, float w, int ventana)
{
    int i, j, k;
    double M, M_0, E0, E, v, cosv, sinv, ee, t_0;
    float cosE;
    float P_sec = P * DAY_SEC;

    //*************************************************************************
    // Toma del array t[] del archivo con la curva modelada con ruido
    //*************************************************************************
    serieTemporal serie = DataRead("data_curve_mod.dat");
    int N = serie.size;

    //*************************************************************************
    // Generación de la RV del ajuste (sin ruido), obtención del ruido-señal
    //*************************************************************************
    double rvFit[N], t_sec[N], noiSig[N];
    ee = 1 - (e * e);
    t_0 = serie.t[0] * DAY_SEC;

    for (i = 0; i < N; i++)
    {
t_sec[i] = serie.t[i] * DAY_SEC;

// Ecuacion para la anomalia media
M = (TWOPI / P_sec) * (t_sec[i] - t_0); // t_0 momento de paso por el perecentro
```

```
// Solucion iterativa de la ecuación de Kepler para la anomalia excentrica
E0 = M;
E = M + e * sin(E0);
while (abs(E - E0) > 0.000001)
{
E0 = E;
E = M + e * sin(E0);
}

cosE = cos(E);
// Ecuaciones para la anomalia verdadera
cosv = (cosE - e) / (1 - e * cosE);   // cosv=0.5=> v=60,-60
sinv = (sqrt(ee) * sin(E)) / (1 - e * cosE);

if (sinv > 0) v = acos(cosv);
else v = -acos(cosv);

// Ecuacion para la RV
rvFit[i] = gamma + K_1 * (cos(w + v) + e * cos(w));   // RV con los parámetros del EXOFIT, sin ruido

// RV con ruido menos RV sin ruido
noiSig[i] = serie.rv[i] - rvFit[i];
    }

    //************************************************************************
    // Escritura en archivo
    //************************************************************************
    FILE *ofile = fopen("data_curve_noisig.dat", "w");
    for (i = 0; i < N; i++)  fprintf (ofile, "%lf\t%lf\t%lf \n", serie.t[i], noiSig[i], rvFit[i]);
    fclose (ofile);

    //************************************************************************
    // Toma del espectro del ruido
    //************************************************************************
    Powerspectrum ("data_curve_noisig.dat", 0);
    Powerspectrum_lomb (2);
    //************************************************************************
    // Suavizado del espectro del ruido
    //************************************************************************
    //Cma ("data_power_noisig.dat", ventana);

    //************************************************************************
    // Cálculo de la frecuencia de Nyquist (fNyq) desde la serie modelada
    //************************************************************************
    //~ double deltaT[N];
    //~ double deltaTmin = serie.t[1] - serie.t[0];
    //~ for (i = 1; i < N; i++)
    //~ {
//~ deltaT[i] = serie.t[i] - serie.t[i-1];
//~ if (deltaT[i] < deltaTmin)  deltaTmin = deltaT[i];
    //~ }
    double deltaTAve = (serie.t[N-1] - serie.t[0])/N;
    double fNyq = 1.0 / (2.0 * deltaTAve);
    fprintf (stdout,"Fnyq = %lf\t \nDeltaTAve = %lf \n", fNyq, deltaTAve);

    //************************************************************************
    // Lectura del espectro del ruido, dejamos las frecuencias hasta la fNyq
    //************************************************************************
    //~ serieTemporal serieEspectroRuido = DataRead("data_power_noisig.dat");
    //~ int N2 = serieEspectroRuido.size;
//~
    //~ FILE *ofile2 = fopen("data_power_noisig_fnyq.dat", "w");
    //~ for (i = 0; i <N2 ; i++)
    //~ {
//~ if (serieEspectroRuido.t[i] < fNyq)  fprintf (ofile2, "%lf\t%lf \n", serieEspectroRuido.t[i], serieEspectroRuido.rv[i]);
    //~ }
    //~ fclose (ofile2);

    return 0;
}


/*************************************************************************
 * FUNCIÓN: RnFit
 *************************************************************************
 * Función para el ajuste del ruido rojo. Llama a una rutina (que a su vez
 * llama a una función) de MATLAB para hacer el ajuste de los parámetros del
 * espectro del ruido rojo
 *************************************************************************
 * No recibe argumentos
 *************************************************************************
 * Devuelve 0 si sale correctamente
```

```
 ***************************************************************************/
int RnFit (void)
{
    system ("matlab -r 'matlab_fit' -nosplash -nodesktop");
    return 0;
}


/***************************************************************************
 * FUNCIÓN: RnTest
 ***************************************************************************
 * Función para calcular el test de significación de los picos en el espectro
 * de la RV. Esta funciona para frecuencias individuales
 ***************************************************************************
 * Argumentos:
 * Rho parámetro del ruido rojo. está relacionado con su memoria, depende
 *  de tau
 * A parámetro del ruido rojo. factor relacionado con su amplitud,
 *  depende de noise
 * fNyq frecuencia de Nyquist (sampleado) en la señal de la RV modelada
 * alpha incertidumbre para el test de significación
 ***************************************************************************
 * Devuelve una estructura que contiene 3 arrays, uno para la frecuencia, otro
 * para la potencia del espectro teórico y otro para el límite de significación
 * También escribe estos arrays al archivo "data_power_noisig_test.dat"
 ***************************************************************************/
serieTemporal RnTest (float Rho, float A, float fNyq, float alpha)
{
    //*************************************************************************
    // Lectura del espectro (hasta fnyq)
    //*************************************************************************
    serieTemporal serieEspectroRuido = DataRead("data_power_noisig_fnyq.dat");
    int N = serieEspectroRuido.size;

    int i;
    double espectroTeorico[N];
    double C_k[N];
    int nu = 2;
    float p = 1 - alpha;
    double chiSq = -nu * log(1 - p);
    float RhoSq = Rho * Rho;
    serieTemporal serieRuidoTest; // serieTemporal que devuelve esta función
    //*************************************************************************
    // Cálculo del espectro teórico y del test
    //*************************************************************************
    for (i = 0; i <N ; i++)
    {
// espectro teórico del ruido rojo
espectroTeorico[i] = A / (1 + RhoSq - 2 * Rho * cos (PI * serieEspectroRuido.t[i] / fNyq));
// test de confiabilidad
C_k[i] = (espectroTeorico[i] / nu) * chiSq;

serieRuidoTest.t[i] = serieEspectroRuido.t[i];
serieRuidoTest.rv[i] = espectroTeorico[i];
serieRuidoTest.other[i] = C_k[i];
    }
    //*************************************************************************
    // Escritura en archivo
    //*************************************************************************
    FILE *ofile = fopen("data_power_noisig_test.dat", "w");
    for (i = 0; i <N ; i++)
    {
if (serieEspectroRuido.t[i] < fNyq)
{
    //fprintf (ofile, "%lf\t%lf\t%lf \n", serieEspectroRuido.t[i], espectroTeorico[i], C_k[i]);
    fprintf (ofile, "%lf\t%lf\t%lf \n", serieRuidoTest.t[i], serieRuidoTest.rv[i], serieRuidoTest.other[i]);
}
    }
    fclose (ofile);
    return serieRuidoTest;
}


/***************************************************************************
 * FUNCIÓN: RnTestMultiple
 ***************************************************************************
 * Función para calcular el test de significación de los picos en el espectro
 * de la RV. Esta es la versión para un testo múltiple
 ***************************************************************************
 * Argumentos:
 * Rho parámetro del ruido rojo. está relacionado con su memoria, depende
 *  de tau
 * A parámetro del ruido rojo. factor relacionado con su amplitud,
```

```
 *  depende de noise
 * fNyq frecuencia de Nyquist (sampleado) en la señal de la RV modelada
 * alpha incertidumbre para el test de significación
 ****************************************************************************
 * Devuelve una estructura que contiene 3 arrays, uno para la frecuencia, otro
 * para la potencia del espectro teórico y otro para el límite de significación
 * También escribe estos arrays al archivo "data_power_noisig_test.dat"
 ****************************************************************************/
serieTemporal RnTestMultiple (float Rho, float A, float fNyq, float alpha)
{
    //**************************************************************************
    // Lectura del espectro (hasta fnyq)
    //**************************************************************************
    serieTemporal serieEspectroRuido = DataRead("data_power_noisig_fnyq.dat");
    int N = serieEspectroRuido.size;
    //**************************************************************************
    // Lectura de la RV para tomar el número de puntos
    //**************************************************************************
    serieTemporal serieRV = DataRead("data_curve_mod.dat");
    int M = serieRV.size;
    //**************************************************************************
    int i;
    double espectroTeorico[N];
    double C_k[N];
    int nu = 2;
    float p = 1 - (alpha / M);
    double chiSq = -nu * log(1 - p);
    float RhoSq = Rho * Rho;
    serieTemporal serieRuidoTest; // serieTemporal que devuelve esta función
    //**************************************************************************
    // Cálculo del espectro teórico y del test
    //**************************************************************************
    for (i = 0; i <N ; i++)
    {
// espectro teórico del ruido rojo
espectroTeorico[i] = A / (1 + RhoSq - 2 * Rho * cos (PI * serieEspectroRuido.t[i] / fNyq));
// test de confiabilidad
C_k[i] = (espectroTeorico[i] / nu) * chiSq;

serieRuidoTest.t[i] = serieEspectroRuido.t[i]; // frecuencias
serieRuidoTest.rv[i] = espectroTeorico[i]; // espectro teórico
serieRuidoTest.other[i] = C_k[i]; // significación
    }
    //**************************************************************************
    // Escritura en archivo
    //**************************************************************************
    FILE *ofile = fopen("data_power_noisig_test.dat", "w");
    for (i = 0; i <N ; i++)
    {
if (serieEspectroRuido.t[i] < fNyq)
{
    //fprintf (ofile, "%lf\t%lf\t%lf \n", serieEspectroRuido.t[i], espectroTeorico[i], C_k[i]);
    fprintf (ofile, "%lf\t%lf\t%lf \n", serieRuidoTest.t[i], serieRuidoTest.rv[i], serieRuidoTest.other[i]);
}
    }
    fclose (ofile);
    return serieRuidoTest;
}
```

## sinusoid_curve.c

```
#include "rn.h"

/****************************************************************************
 * FUNCIÓN: SinCurve
 ****************************************************************************
 * Función para generar una sinusoide como primera aproximación a una RV. Llama
 * a la función de generación de ruido
 ****************************************************************************
 * Argumentos:
 * t_0 tiempo inicial de la serie temporal
 * t_f tiempo final de la serie temporal
 * P período orbital del planeta
 * N número de puntos de la curva
 * K_1 semiamplitud de la sinusoide
 * gamma RV_0, velocidad sistémica o velocidad radial del sistema (centro de
 *  masa)
 * phi desfase de la sinusoide
 * noise amplitud del ruido
 * tau parámetro de autocorrelación (memoria) del ruido rojo (proceso AR1)
 *  se usa en el caso de querer generar ruido rojo
 * arg argumento switch para escoger el tipo de ruido a agregar
```

```
 ***************************************************************************
 * Devuelve 0 si finaliza correctamente
 ***************************************************************************/
int SinCurve (double t_0, double t_f, float P, int N, float K_1, float gamma, float phi, float noise, float tau, int arg)
{
int i, j, k;
double t[N], rv[N], seed, swap;
double w = TWOPI/P;
float gamma_ms = gamma * 1000.0;
float eta[N];
double phi_d = phi * PI / 180;

//***************************************************************************
// Generamos el t aleatoriamente y generamos rv
//***************************************************************************
// Inicializacion de la semilla y los aleatorios
seed = time(NULL) % 30000;
RandomInitialise (seed, seed+100);
// Generamos t
for (i = 0; i < N; i++)
{
t[i] = t_0 + RandomUniform() * (t_f - t_0);
}
//***************************************************************************
// Algoritmo Bubble sort para ordenar el array de tiempo
//***************************************************************************
for (i = 0; i < N - 1; i++)
{
for (j = 0 ; j < N - i - 1; j++)
{
if (t[j] > t[j+1])  // descendente, usar: <
{
swap = t[j];
t[j] = t[j+1];
t[j+1] = swap;
}
}
}
for (i = 0; i < N; i++)
{
// Generamos RV sin ruido
rv[i] = gamma_ms + K_1 * cos(w * t[i] + phi_d);
}
    //***************************************************************************
// Llamada a la función de generación de ruido
    //***************************************************************************
    if (arg == 1)                                                          // Para ruido no Gaussiano
    {
        Noise (arg, N, noise, tau, t, rv);
    }
    if (arg == 2)                                                          // Para ruido blanco Gaussiano
    {
        Noise (arg, N, noise, tau, t, rv);
    }
    if (arg == 3)                                                          // Para ruido rojo
    {
        Noise (arg, N, noise, tau, t, rv);
    }

return 0;
}
```

## varlib.c

```
#include "rn.h"

/***************************************************************************
 * FUNCIÕN: GausTest
 ***************************************************************************
 * Función para testear la "gaussianidad" de los generadores de números pseudo-
 * aleatorios. Genera una distribución gaussiana con una media y desviación
 * estandar dadas de antemano, y luego lo comprueba
 ***************************************************************************
 * Argumentos:
 * mean media
 * stddev desviación estándar
 ***************************************************************************
 * Esta función no devuelve valores, escribe en archivo "data_gaustest.dat"
 ***************************************************************************/
void GausTest (float mean, float stddev)
{
    int ene = 1000;
    int eme = 10000;
```

```
    int i;
    double r, bins[ene+1];
    double sum=0,sum2=0;
    FILE *ofile;
    ofile = fopen("data_gaustest.dat", "w");

    for (i=0;i<ene;i++)
    bins[i] = 0;

    RandomInitialise(1802,9373);
    for (i = 0; i < eme; i++)
    {
    r = RandomGaussian(mean, stddev);
    //r = (RandomUniform()-0.5) * 2.0;
    sum += r;
    sum2 += (r*r);

    //~ if (r < -4)
     //~ r = -4;
    //~ if (r > 4)
     //~ r = 4;

    fprintf(ofile, "%d\t%g\n", i, r);

    bins[(int)(ene/2 + r * (ene/2) / 4.0)]++;
    }
    fprintf(stderr,"  emeedia = %g\n",sum / eme);
    fprintf(stderr,"  Deviación Estandar = %g\n",sqrt((sum2 - sum*sum/eme)/eme));

//   for (i=0;i<ene;i++)
//       printf("%g %g\n",i/(double)ene,bins[i]);
    fclose(ofile);
}


/*****************************************************************************
 * FUNCIÓN: Cma
 *****************************************************************************
 * Función con algoritmo de suavizado "central moving average"
 *****************************************************************************
 * Argumentos:
 * file nombre del archivo con la serie a suavizar
 * ventana ventana de suavizado de la señal
 *****************************************************************************
 * Devuelve 0 si sale correctamente
 *****************************************************************************/
int Cma (char file[], int ventana)
{
    int i, j ,k;
    //*************************************************************************
    // Lectura del archivo con la serie temporal
    //*************************************************************************
    serieTemporal serieRuidoEspectro = DataRead(file);
    int M = serieRuidoEspectro.size;

    //*************************************************************************
    // Suavizado (Smoothing) - CENTRAL MOVING AVERAGE de un array
    //*************************************************************************
    double *array = serieRuidoEspectro.rv; // espectro del ruido no rv
    int ala = floor(ventana / 2);
    double arrayNum[M], arraySmooth[M];

    for (i = ala; i < M-ala; i++) // posiciones intermedias
    {
arrayNum[i] = array[i];
for (j = 1; j <= ala; j++)  arrayNum[i] += array[i + j] + array[i - j];
arraySmooth[i] = arrayNum[i] / ventana;
    }
    for (i = 0; i < ala; i++)  arraySmooth[i] = arraySmooth[ala];            // posiciones iniciales, hasta el tamaño del ala
    for (i = M-ala; i < M; i++)  arraySmooth[i] = arraySmooth[M-ala-1];      // posiciones finales, N - ala

    // Salvado de archivo "data_power_noisig.dat"
    FILE *ofile;
    ofile = fopen("data_power_noisig.dat", "w");
    for (i = 0; i < M; i++)
    {
//fprintf (stdout,"%lf\t%lf\n", serieRuidoEspectro.t[i], arraySmooth[i]);
fprintf (ofile,"%lf\t%lf\n", serieRuidoEspectro.t[i], arraySmooth[i]);
    }
    fclose (ofile);
    return 0;
}
```

```
/*****************************************************************************
 * FUNCIÓN: Cma2
 *****************************************************************************
 * Función con algoritmo de suavizado "central moving average" otra versión
 * donde se calculan diferente los extremos
 *****************************************************************************
 * Argumentos:
 * file nombre del archivo con la serie a suavizar
 * ventana ventana de suavizado de la señal
 *****************************************************************************
 * Devuelve 0 si sale correctamente
 *****************************************************************************/
int Cma2 (char file[], int ventana)
{
    int i, j ,k;
    //*************************************************************************
    // Lectura del archivo con la serie temporal
    //*************************************************************************
    serieTemporal serieRuidoEspectro = DataRead(file);
    int M = serieRuidoEspectro.size;

    //*************************************************************************
    // Suavizado (Smoothing) - CENTRAL MOVING AVERAGE versión 2, con peso
    // mayor para los extremos
    //*************************************************************************
    double *array = serieRuidoEspectro.rv;
    int ala = floor(ventana / 2);
    double arrayNum[M], arraySmooth[M];
    for (i = 1; i < ala; i++)         // posiciones hasta el tamaño del ala
    {
arrayNum[0] = array[0];         // primera posición
arrayNum[i] = array[i];
for (j = 1; j <= ala; j++)
{
    arrayNum[0] += array[j];
    arrayNum[i] += array[i + j];
    if (i - j >= 0)  arrayNum[i] += array[i - j];
}
    arraySmooth[0] = arrayNum[0] / (ventana - ala);
    arraySmooth[i] = arrayNum[i] / (ventana - ala + i);
    }
    for (i = ala; i < M - ala; i++)         // resto de posiciones del array
    {
arrayNum[i] = array[i];
for (j = 1; j <= ala; j++)  arrayNum[i] += array[i + j] + array[i - j];
arraySmooth[i] = arrayNum[i] / ventana;
    }
    k = ala - 1;
    for (i = M - ala; i < M-1; i++)         // posiciones N - ala
    {
arrayNum[i] = array[i];
for (j = 1; j <= ala; j++)
{
    arrayNum[i] += array[i - j];
    if (i + j <= M)  arrayNum[i] += array[i + j];
}
arraySmooth[i] = arrayNum[i] / (ventana - ala + k);
k--;
    }
    arrayNum[M-1] = array[M-1];              // última posición
    for (j = 1; j <= ala; j++)  arrayNum[M-1] += array[M-1-j];
    arraySmooth[M-1] = arrayNum[M-1] / (ventana - ala);

    // Salvado de archivo "data_power_noisig.dat"
    FILE *ofile;
    ofile = fopen("data_power_noisig.dat", "w");
    for (i = 0; i < M; i++)
    {
//fprintf (stdout,"%lf\t%lf\n", serieRuidoEspectro.t[i], arraySmooth[i]);
fprintf (ofile,"%lf\t%lf\n", serieRuidoEspectro.t[i], arraySmooth[i]);
    }
    fclose (ofile);
    return 0;

}
```