Universidad Nacional de Córdoba
Facultad de Matemática, Astronomía, Física y Computación

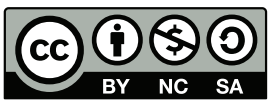
Desarrollo e Implementación de Algoritmos para QGIS en Análisis de Series de Tiempo

Trabajo final de la Licenciatura en Ciencias de la Computación

Autor: Natalia Bortagaray

Directora de Tesis: Silvia María Ojeda

Colaborador: Marcos Alejandro Landi



Córdoba, Argentina
2018

Resumen

Los sensores remotos han sido valiosos auxiliares de los ecólogos en las últimas décadas. El uso más frecuente de estas herramientas ha sido la caracterización estructural del paisaje. En general, se ha recurrido a la interpretación visual y clasificación digital de imágenes. Si bien el uso de la teledetección ha permitido avances importantes en Ecología, parece estar por debajo de su potencial.[31]

En este trabajo se desarrollarán e implementarán rutinas computacionales útiles para el procesamiento y análisis de series de tiempo de Índice de Vegetación Normalizado (Normalized Difference Vegetation Index, NDVI) extraídas de imágenes obtenidas por el sensor MODIS (Moderate Resolution Imaging Spectroradiometer), los datos producidos como resultado de nuestros programas pueden utilizarse, por ejemplo, para analizar y comparar la vegetación de distintos puntos geográficos a través del tiempo.

La propuesta se ejecutará en forma de plugins, escritos en lenguaje Python, como aporte para el sistema de información geográfica de código libre QGIS.[16]

Abstract

Over the last decades, remote sensors provided a valuable service to ecologists. The most frequent use of this tool has been the structural characterization of landscape. Generally visual interpretation and classification have been the most common approaches. Even though this use of remote sensing allowed important advances in ecology, it seems to be below its full potential.[31]

In this work we will develop and implement useful computational rutines to process and analyze time series of Normalizad Difference Vegetation Index (NDVI) extracted from images of the Moderate Resolution Imaging Spectroradiometer (MODIS) sensor, resulting data can be used, for example, to analyze and compare vegetation of different geographical points through time.

The proposal will be executed in form of plugins, write in Python lenguage, as contribution for QGIS an open source geographic information system.[16]

Índice general

Capítulo 1

Introducción

La observación remota de la superficie terrestre constituye el marco de estudio de la teledetección, donde no solo nos referimos a los procesos que permiten obtener una imagen desde el aire o el espacio, sino también su posterior tratamiento, en el contexto de una determinada aplicación.[26]

Los primeros satélites de observación terrestre comenzaron a operar a principios de la década de 1970. Desde entonces el sistema científico-técnico ha puesto a disposición de la sociedad herramientas y modelos conceptuales que traducen los datos registrados por los sensores remotos en conocimiento útil para la toma de decisiones en múltiples áreas incluyendo las ciencias ambientales y la conservación.

Los sensores a bordo de satélites registran energía electromagnética emitida o reflejada por un objeto o superficie en distintas bandas del espectro electromagnético. Las imágenes satelitales proveen entonces datos cuantitativos y espacialmente continuos de la superficie y, en tal sentido, son mucho más que una fotografía.[31]

Al disponer de una gran masa de datos sobre el territorio, el uso de computadoras para procesar los mismos es inevitable, ampliando la capacidad de análisis e interpretación del problema a estudiar, permitiendo poner énfasis sobre el planteamiento de modelos y la observación de los resultados.[26]

Los sensores remotos han sido valiosos auxiliares de los ecólogos en las últimas décadas, el uso más frecuente ha sido la caracterización estructural del paisaje. En general, se ha recurrido a la interpretación visual y la clasificación digital de imágenes. Si bien este uso de la teledetección ha permitido avances importantes en Ecología, parece estar por debajo de su potencial.[31]

Para obtener información confiable sobre la respuesta temporal del funcionamiento de los ecosistemas ante cambios ambientales, es necesario contar con plataformas satelitales de elevada resolución temporal con varias imágenes diarias, si bien este tipo de resolución temporal ya estaba presente en el sensor Scanning Radiometers a bordo de la plataforma satelital TIROS-M (Television Infrared Observation Satellite)[9] en 1970, su baja resolución espacial (8x8 km) dificultaba su uso para estudios detallados sobre el funcionamiento del ecosiste-

ma.Esta situación cambió drásticamente con el lanzamiento de las plataformas Terra (1999) y Aqua (2002), los cuales llevan a bordo el sensor MODIS (Moderate Resolution Imaging Spectroradiometer)[7]. El mismo cuenta con resolución que va desde los 250 x 250 m hasta 1 x 1 km según el canal de observación, a la vez que el funcionamiento en tándem de las plataformas Terra y Aqua permiten obtener hasta 4 imágenes diarias, lo cual transformó al sensor MODIS en una herramienta ideal para observar el funcionamiento de los ecosistemas tanto a escala global como local, desde esa fecha se ha incrementado en forma constante la cantidad de estudios destinados a monitorear el estado y evolución del funcionamiento del ecosistema.[22][30]

1.1. Objetivos

1.1.1. Objetivos generales

- Este trabajo tiene como objetivo desarrollar distintos plugins que permitan a los usuarios poder pre-procesar y analizar series de tiempo de NDVI obtenidas a partir de imágenes captadas por el sensor MODIS sobre un mismo sitio geográfico.
- Realizar un aporte de código libre para el uso de series de tiempo de NDVI para el sistema de información geográfico QGIS.

1.1.2. Objetivos específicos

- Crear un plugin de identificación y eliminación de datos anómalos en series de tiempo de NDVI.
- Crear un plugin de filtrado de series de tiempo de NDVI, el cual implemente el filtro de Savitzky-Golay y el filtro Doble Gaussiano.
- Crear un plugin de extracción de características fenológicas de las series de tiempo de NDVI.

1.2. Organización del trabajo

El trabajo esta dividido en tres etapas:

- **Primera etapa:** Se de desarrollarán algoritmos para el pre-procesamiento de las imágenes, los que permitirán tanto la detección e interpolación de datos anómalos, como el filtrado de las series de tiempo (Filtros: Savitzky-Golay y Doble Logístico).
- **Segunda Etapa:** Se implementarán rutinas que nos permitirán la extracción de las características de las series generadas en la primera etapa, tales

como máximos, mínimos, que facilitarán al usuario de la herramienta analizar la serie temporal, compararla con otras series y sacar conclusiones sobre la misma.

- **Tercera Etapa**: Comparación con otras herramientas similares de procesamiento de imágenes, pero de código cerrado; se presentan también conclusiones y trabajos a futuro.

Capítulo 2

Marco Teórico

En las siguientes secciones se desarrollarán conceptos teóricos necesarios para comprender el problema a tratar en este trabajo.

2.1. Moderate-Resolution Imaging Spectroradiometer (MODIS)

MODIS es un instrumento científico lanzado en orbita terrestre por la NASA en 1999 a bordo del satélite Terra y en 2002 a bordo del satélite Aqua; Terra órbita alrededor de la Tierra de manera tal que pasa de norte a sur a través del ecuador en la mañana, mientras que Aqua pasa de sur a norte a través del ecuador por la tarde, adquiriendo datos en 36 bandas espectrales.

Tanto el sensor MODIS-Terra como el sensor MODIS-Aqua escanean la totalidad de la superficie terrestre cada 1 o 2 días, el trabajo en conjunto de ambos sensores permite optimizar la obtención de imágenes libre de nubes, y minimizar el efecto óptico de sombras y reflejos que ocurren al amanecer y atardecer.

Estos datos ayudan a entender la dinámica global y los procesos que ocurren sobre la superficie terrestre, océanos y en parte de la atmósfera, jugando un rol importante en la toma de decisiones con respecto al medio ambiente.[8][7]

MODIS obtiene información sobre:

- Temperatura de superficie (suelo y océano).
- Color del océano (sedimentos, fitoplancton).
- **Cartografía de la vegetación global**.
- Características de la nubosidad.
- Concentraciones de aerosoles.

En nuestro trabajo nos centraremos en la cartografía de la vegetación, analizando series de tiempo de **"índice de vegetación de diferencia normalizada"** (NDVI) extraídas de imágenes tomadas por MODIS sobre un mismo lugar; en la siguiente sección explicaremos detalladamente este indice de vegetación.

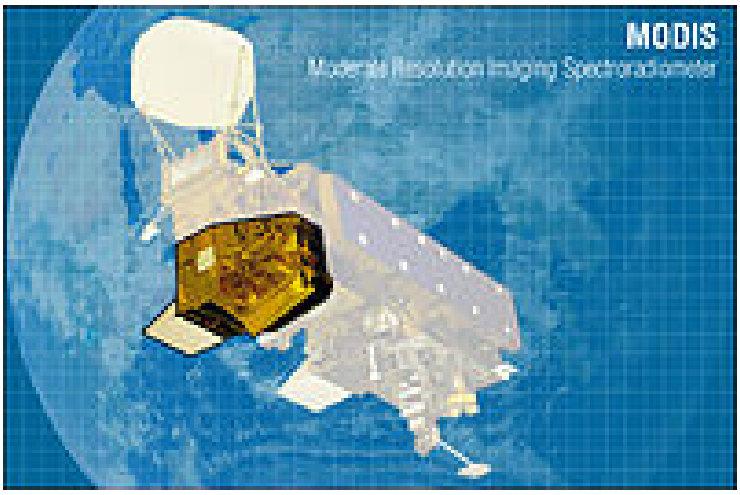


Figura 2.1: MODIS a bordo de Aqua.[8]

2.2. Indices de Vegetación

Los sensores a bordo de las plataformas satelitales registran la energía electromagnética emitida o reflejada por la superficie terrestre en diferentes bandas del espectro electromagnético.[34]; sin embargo, para que los datos registrados por el sensor a bordo del satélite se transformen en información útil para la sociedad, es necesario incorporar la misma en modelos teóricos; en este trabajo, interesa interpretarla en términos biológicos[31], por lo cual es necesario conocer como la energía electromagnética captada por el sensor interactúa con la vegetación para luego incorporar dicha información en un modelo conceptual.

Una de las formas más extendidas y sencillas de modelar los datos captados por el satélite es mediante la construcción de **"Índices de Vegetación"** (IV).

La gran mayoría de los índices de vegetación se basan en la interacción de luz roja e infrarroja con el proceso de fotosíntesis de las plantas; los pigmentos fotosintéticos absorben la mayor parte de la luz roja (0,6 a 0,7 μm) que reciben, mientras que reflejan la mayor parte de la luz en el infrarrojo cercano (0,7 a 1,1 μm).(Figura 2.2).

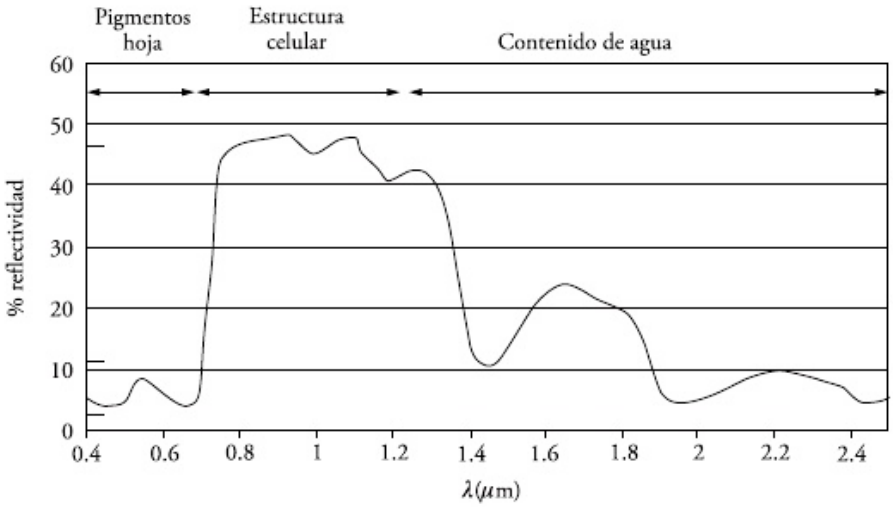


Figura 2.2: Respuesta espectral típica de la vegetación ante las diferentes longitudes de onda de la luz incidente.[34]

2.2.1. Indice NDVI

Uno de los índices de vegetación más utilizados por la comunidad científico-técnica en el pasado y en la actualidad es el **"Normalized Difference Vegetation Index"** (NDVI) (Ecuación 2.1)[31]; su amplia popularidad se debe principalmente a su relación con parámetros básicos para el monitoreo del ecosistema, como por ejemplo la cantidad de biomasa presente, el índice de área foliar específica y la productividad primaria neta.

$$NDVI = \frac{L_{irc} - L_r}{L_{irc} + L_r} \tag{2.1}$$

Donde:

- L_r: Luz roja (0,6 a 0,7 μm).
- L_{irc}: Luz infrarroja cercana incidente (0,7 a 0,9 μm).

La luz roja y la luz infrarroja cercana incidente son en sí cocientes de la radiación reflejada sobre la radiación, éstos toman valores dentro de un rango de 0,0 a 1,0. El NDVI varía como consecuencia entre -1,0 y +1,0.[12]

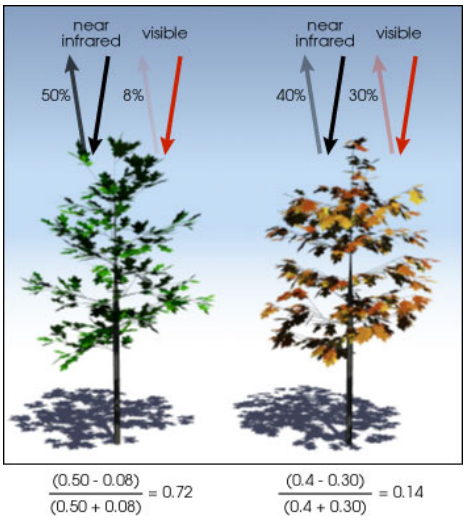


Figura 2.3: Ejemplo del calculo del indice NDVI.[11]

Sin embargo el NDVI posee importantes limitaciones:

1. Presenta saturación de la capacidad de respuesta del índice a elevados niveles de biomasa.

2. No solo es sensible a la actividad fotosintética, sino también a la luz reflejada por el sustrato en el que se halla la vegetación, denominada "Background Signal".

El efecto del "Canopy Background Signal" es particularmente importante en sitios con baja nivel de cobertura vegetal y alto porcentaje de suelo desnudo, lo cual dificulta la comparación de la actividad fotosintética en sitios con diferentes tipos de suelo.[28]

2.2.2. Indice EVI

Debido a los problemas con el NDVI citados en la sección anterior, se propuso el uso del índice **"Enhanced Vegetation Index"** (EVI)[2] (Ecuación 2.2).Este índice posee la ventaja de que es menos sensible a los efectos del "Canopy Background Signal", no presenta saturación de la capacidad de respuesta a elevados niveles de biomasa y además es menos sensible al efecto de las condiciones atmosféricas imperantes durante la observación.

Por sus ventajas sobre el indice NDVI, el uso de EVI se esta inponiendo en la comunidad científico-técnica para el monitoreo del estado y evolución de los ecosistemas.

$$EVI = \frac{G * L_{irc} - L_r}{L_{irc} + C1 * L_r - C2 * L_a + L} \tag{2.2}$$

Donde:

- L_r: Luz roja (0,6 a 0,7 μm).
- L_{irc}: Luz infrarroja cercana incidente (0,7 a 0,9 μm).
- L_a: Luz azul (0,4 a 0,5 μm).
- $C1 = 6$: Coeficiente modelador de la resistencia atmosférica de la luz roja.
- $C2 = 7,5$: Coeficiente modelador de la resistencia atmosférica de la luz infrarroja.
- $L = 1$: Coeficiente de ajuste del "Canopy Background Signal".
- $G = 2,5$: Factor de ganancia.

Sin embargo, en nuestro trabajo utilizaremos el NDVI, ya que si bien presenta desventajas con respecto al EVI, las imágenes obtenidas son del primero de los indices.

2.3. Fenología satelital y series de tiempo

La fenología de las plantas estudia el ciclo vegetativo de las mismas a lo largo del año; su estudio es de gran utilidad para comprender la respuesta de las diferentes clases de vegetación ante las variaciones climáticas.

Los indices de vegetación (IV) se hallan fuertemente relacionados con la cantidad de biomasa fotosintéticamente activa presente en el ecosistema, por lo tanto la serie temporal de datos obtenida a lo largo del ciclo anual refleja el ciclo anual de la vegetación.

A través de la obtención de parámetros descriptivos de la serie de tiempo anual de IV es posible obtener las características fenológicas de la vegetación (Figura 2.4).

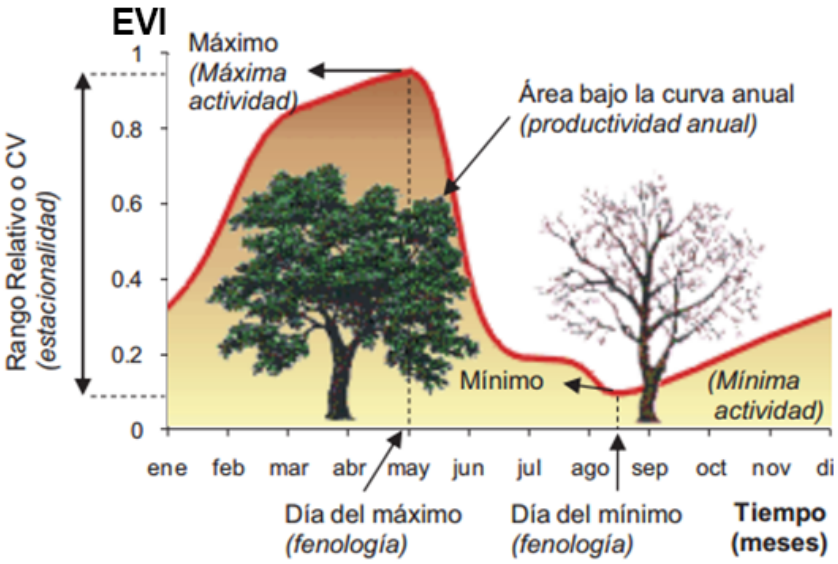


Figura 2.4: Relación entre el ciclo anual de la vegetación y las características descriptivas de este ciclo obtenidas a partir de una serie de tiempo de EVI.[24]

A través del estudio de estas características fenológicas, también llamadas fenométricas satelitales, no es posible distinguir algunas etapas cruciales del ciclo de vida de las plantas, como por ejemplo floración o formación de frutos y semillas. Sin embargo, a través del estudio de las mismas es posible detectar y cuantificar la fecha y la magnitud de los eventos relacionados a la captación de carbono y generación de biomasa.[32]

Algunos parámetros a extraer serían, por ejemplo, la fecha del pico de la actividad fotosintética, las fechas del inicio y finalización de la estación de crecimiento, su duración, la magnitud del nivel pico y mínimo de actividad fotosintética y la magnitud del rango relativo anual de la vegetación.

En nuestro caso obtendremos series de tiempo a través de un conjunto de imágenes de NDVI obtenidas por MODIS en un mismo lugar como se muestra en la figura 2.5

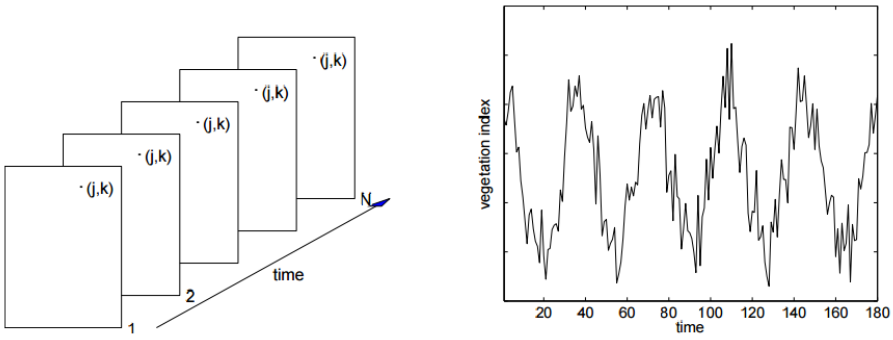


Figura 2.5: Los indices de vegetación se organizan en imágenes (Figura izquierda). Una imagen i contiene los indices de vegetación de un lugar geográfico en el tiempo t_i, si extraemos los valores del pixel (j, k) de manera ordenada para los tiempos consecutivos, se obtiene una serie de tiempo de indices de vegetación para ese pixel (Figura derecha).[27]

2.4. Fuentes de error en la formación de imágenes satelitales

Para incorporar los datos obtenidos por las diferentes plataformas satelitales en modelos teóricos y así comprender los procesos que ocurren en el medio ambiente, es necesario que los datos tengan un significado definido y univoco.Este requisito es indispensable para poder interpretar y comparar los datos obtenidos por diferentes sensores, en distintas fechas y regiones del espectro electromagnético (bandas). Sin embargo, durante el proceso de formación de la imagen satelital existen múltiples fuentes de errores que generan distorsiones en el producto obtenido, las cuales tienen que ser corregidas o al menos tratadas.

2.4.1. Efecto de atmosféricos

Uno de los fenómenos que produce distorsiones en las imágenes satelitales es el llamado **Efecto de atmosféricos**, la luz solar debe atravesar e interactuar con la atmósfera para iluminar la escena en la superficie de la Tierra y luego volver a atravesarla para llegar hasta el sensor a bordo del satélite; en su paso por la atmósfera la luz sufre diversos procesos que modifican tanto la luz que llega hasta la escena como la luz que llega hasta el sensor a bordo del satélite, a medida que la luz atraviesa la atmósfera es absorbida por los diferentes gases que la componen, debido a esto el 20 % de la energía lumínica es absorbida por las partículas atmosféricas y no llega hasta el suelo.[29]La luz que no es absorbida por los gases atmosféricos tiende a interactuar con las partículas de la atmósfera, las que dispersan los rayos de luz alterando su curso, por lo cual se incrementa la cantidad de luz que llega hasta el suelo en forma de radiación

difusa.

La dispersión juega un papel importante en la formación de la imagen, ya que genera que luz que llega hasta las celdas del sensor esté mezclada con la luz que debería llegar a los pixeles vecinos.

Los principales causantes del fenómeno de dispersión son los aerosoles en suspensión y el vapor de agua. La presencia de nubes durante el momento de formación de las imágenes representa un tipo de dispersión no selectivo, debido a que las gotas de agua dispersan la luz incidente en todas las direcciones impidiendo observar la escena que se halla debajo (Figura 2.6).[26]

Figura 2.6: Fenómeno de Dispersión dentro de la atmósfera modificando la trayectoria de los rayos luminosos.[21][10]

2.4.2. Efecto del ángulo de incidencia solar

Los satélites más utilizados por la comunidad científica-técnica a nivel mundial, como por ejemplo las misiones Landsat, Sentinel, MODIS, AVHRR, CBERS y SPOT entre otras, son de tipo heliosíncronos, lo que significa que pasan siempre a la misma hora para obtener una imagen de un determinado lugar de la tierra. Por lo tanto la cantidad de luz que recibe la escena posee un amplio rango de variación a lo largo del año, debido a que el ángulo de elevación solar presenta fuertes variaciones. Es decir: que cuando el sol se halla en el cenit en el cielo, la cantidad de luz incidente que se distribuye sobre una superficie de terreno es más pequeña que cuando los rayos caen en forma oblicua.

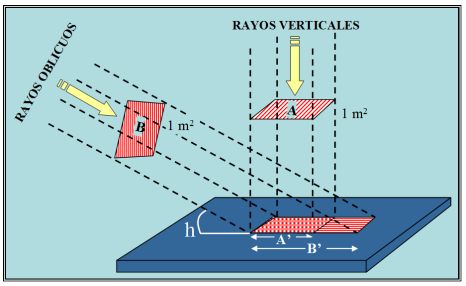


Figura 2.7: Efecto del angulo de incidencia solar.[6]

Por lo tanto las imágenes captadas en verano tendrán un ángulo de elevación solar mayor que las captadas en invierno, y en consecuencia la recibirán mayor cantidad de luz por unidad de superficie.

2.4.3. Efectos del sustrato

Además de los efectos atmosféricos, la manera en que la luz interactúa con la escena genera distorsiones en el proceso de formación de la imagen.

Un supuesto importante del proceso de formación de imágenes satelitales, es que la superficie sobre la cual incide la luz posee un comportamiento lambertiano, esto significa que la luz luego de rebotar contra la superficie se dispersa en todas las direcciones con la misma intensidad; sin embargo, no todas las superficies poseen este comportamiento, como por ejemplo el asfalto o los cuerpos de agua, donde la luz es reflejada únicamente en una dirección; la mayoría de las superficies observadas a través de sensores satelitales poseen un comportamiento intermedio entre estos dos extremos.[36]

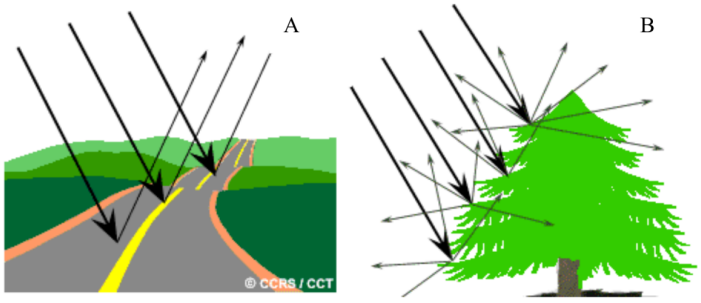


Figura 2.8: Efecto del sustrato.[1]

Por lo cual, si se desea comparar superficies con propiedades reflectivas muy diferentes, es necesario modelar dichas diferencias de comportamiento.

2.4.4. Efecto de sombras topográficas

La presencia de sombras topográficas puede tener un marcado efecto en regiones montañosas, debido a que en las zonas del paisaje que quedan ocultas a los rayos del sol, estos pixeles captados por el sensor poseen un valor cercano a cero.

2.5. Disponibilidad de programas

Actualmente existen diferentes programas libres que permiten obtener algunas de las fenologías satelitales antes descriptas.

El paquete "Phenex"[14] perteneciente al programa estadístico R, desarrollado por Lange y Doktor en 2015, permite pre-procesar y filtrar series de tiempo obtenidas a partir de los sensores AVHRR y MODIS. A su vez, permite modelar y extraer fenométricas satelitales. Sin embargo, a pesar de que el programa R posee capacidades para procesar imágenes, no ha sido diseñado con ese propósito. Por lo tanto, posee capacidades limitadas para el procesamiento, la visualización de los resultados y para trabajar conjuntamente con sistemas de información geográfica. Además, la interfaz de usuario basada en programación por consola a través de líneas de comando ha desalentado al uso por parte de la comunidad científico-técnica.

El programa TIMESAT[20] desarrollado por Eklundh and Jönsson en 2002 es, quizás, el más popular para pre-procesar y modelar series de tiempo de NDVI o EVI y calcular las respectivas fenométricas satelitales (Kuenzer et al. 2015). El mismo se halla implementado en FORTRAN y MATLAB, además puede ser instalado en forma gratuita. Sin embargo presenta varias limitaciones, entre las que se pueden mencionar:

1. El programa ha sido diseñado pensando solamente en la extracción de las fenométricas satelitales. No es posible obtener las imágenes de NDVI o EVI pre-procesadas; la única salida del programa son las fenométricas. Esto representa una gran falencia del programa, teniendo en cuenta el alto tiempo computacional requerido para el pre-procesamiento de las imágenes.

2. No tiene la capacidad de procesar las imágenes MODIS Quality generadas por el grupo de trabajo MODIS Science Team Quality Assessment.

3. No posee rutinas para analizar el comportamiento de la vegetación en cada estación del año (verano, otoño, invierno y primavera).

4. No se halla incorporado como herramienta en un programa de sistema de información geográfica.

5. A pesar de ser un programa de uso gratuito, no es de código abierto ya que los algoritmos se hallan encriptados.

2.6. QGIS

Para trabajar con las imágenes de NDVI anteriormente nombradas utilizaremos un **Sistema de Información Geográfica** (SIG) de código libre, multiplataforma, llamado **QGIS** (anteriormente Quantum Gis).

QGIS nació como un proyecto en Mayo de 2002 y se estableció en SourceForce en junio de ese año. La idea era crear un software SIG (tradicionalmente un software propietario costoso) el cual fuera accesible para cualquiera que tuviera acceso a una computadora personal.

El programa proporciona una interfaz amigable, además de funciones y características comunes, permite manejar formatos raster y vectoriales a través de las bibliotecas GDAL y OGR, así como bases de datos.Se distribuye bajo la Licencia Pública General GNU (GPL), el desarrollo bajo esta licencia significa que se puede revisar y modificar el código fuente y garantiza que siempre se tendrá acceso a un programa de SIG que es libre de costo y puede ser libremente modificado, además permite la integración de plugins desarrollados tanto en C++ como en Python.[16]Estos plugins son los que desarrollaremos en este trabajo, aprovechando así las características que ofrece QGIS.

Elegimos trabajar sobre esta plataforma ya que proporciona facilidades a sus usuarios para la creación de scrips y plugins, esta muy bien documentado y al ser de código abierto otros usuarios podrían hacer cambios sobre la funcionalidad de los mismos según lo necesiten.

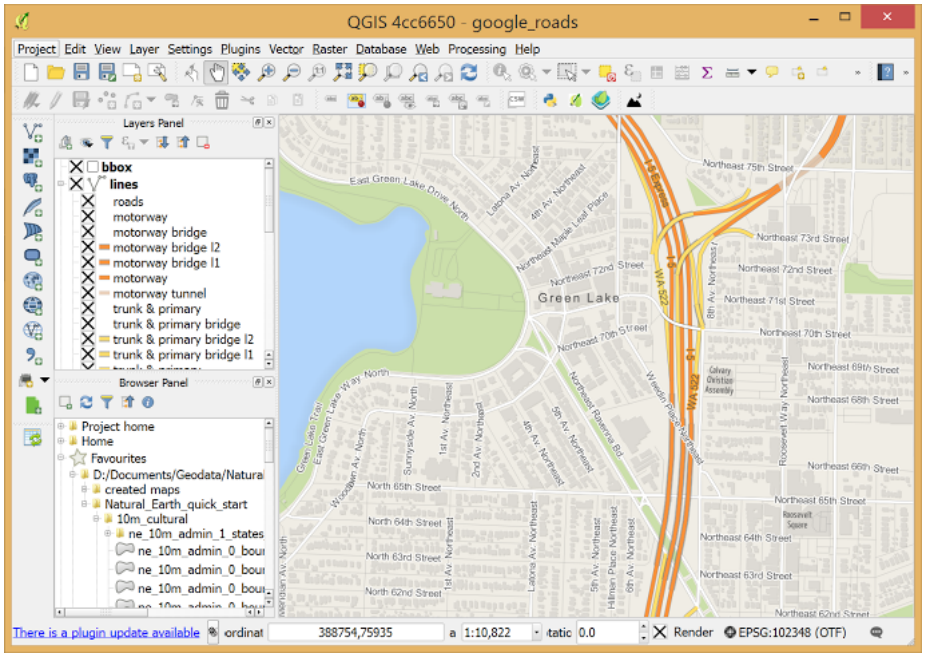


Figura 2.9: Interfaz gráfica de QGIS.

Capítulo 3

Preprocesamiento de imágenes

3.1. Eliminación de datos anómalos

Para la correcta interpretación de la información científico-técnica generada a partir de series de datos de sensores remotos, es necesario poder discriminar entre los artefactos de medición y los verdaderos cambios ocurridos en los procesos terrestres.Para esto es necesario contar con fuentes de información accesorias que permitan evaluar la calidad de los productos utilizados y detectar la presencia de datos que no cumplen con la calidad necesaria para ser utilizados.[33]

Para detectar los datos de baja calidad presentes en las series temporales se utilizaron dos procedimientos:

- El primer procedimiento se basó el uso del producto "MODIS Quality Assessment" que se genera para cada imagen del producto MOD13Q1[37].
- El segundo procedimiento se basó en el método propuesto por Chen [25] para detectar la presencia de datos anómalos a partir del calculo de una curva de tendencia utilizando el filtro polinómico de Savitzky-Golay.

Si bien estos métodos son independientes se recomienda utilizar ambos en forma sucesiva y así detectar la mayor cantidad de datos anómalos posibles.

3.1.1. Producto MODIS Quality Assessment

Cada imagen de índice de vegetación de MODIS (MOD13Q1) posee asociada una imagen del producto Quality Assessment. Luego, el pixel (i, j) de la imagen de calidad tendrá asignado un código numérico de 16 bits, que posee siete campos de información relacionados al proceso de formación de la imagen y dos campos que son el resultado de índices sintéticos de calidad correspondientes al pixel (i, j) de su imagen de NDVI.

Sin embargo, la complejidad del proceso de decodificación de la información contenida en estos datos hace que frecuentemente no se utilicen.

Cada pixel de la imagen Quality Assessment se le asigna un valor entre 0 y 65535. Este valor debe ser convertido a su equivalente número binario de 16 bits (Figura 3.1), luego cada campo de información debe ser descifrado en forma independiente según las pautas establecidas en la siguiente tabla.

El índice de calidad Usefulnessçorrespondiente a los Bits 2 a 5 utiliza información sobre:

1. La cantidad de aerosoles.
2. La realización de la corrección BRDF[3].
3. La realización de la corrección atmosférica.
4. Presencia de nubes.
5. Presencia de sombras geográficas.
6. Ángulo de observación del sensor.
7. Ángulo de elevación solar.

Según estos factores otorga un valor de calidad al pixel, como se muestra en la siguiente tabla:

Bit	Parametro	Valor	Descripción
0-1	Índice Quality (MODLAND QA Bits)	00	IV producido, con buena calidad
		01	IV producido, Chequear otra fuente de información
		10	IV producido, probablemente nublado
		11	Pixel no producido
2-5	Índice Usefulnes	0000	Máxima calidad
		0001	Menor calidad
		0010	Calidad decreciente
		0100	Calidad decreciente
		1000	Calidad decreciente
		1001	Calidad decreciente
		1010	Calidad decreciente
		1100	Calidad minima
		1101	Calidad tan baja que el pixel no es útil
		1110	Datos nivel L1B faltante
		1111	Dato no utilizable o no producido por otras razones
6-7	Cantidad de aerosoles	00	Problemas climatológicos
		01	Mínima
		10	Intermedia
		11	Máxima.
8	Nubes adyacentes detectadas	0	No
		1	Si
9	CorrecciónBRDF	0	No
		1	Si
10	Nubosidad	0	No
		1	Si
11-13	Máscara suelo/agua	000	Océano superficial
		001	Tierra
		010	Costa de océanos y lagos
		011	Aguas continentales superficiales
		100	Cuerpo de agua transitorio
		101	Aguas continentales profundas
		110	Aguas ocáanicas continentales
		111	Océano profundo
14	Posible presencia de Hielo/Nieve	0	No
		1	Si
15	Posible presencia de Sombra	0	No
		1	Si

Cuadro 3.1: Descripción de los campos de información del producto MODIS Quality Assessment. En verde se resaltan los niveles de calidad Usefulnes considerados aceptables.[37]

Actualmente Uselfulness es el índice de calidad más completo producido por los diferentes grupos de trabajo de MODIS, en base a esto se decidió utilizar este índice para detectar la presencia de datos que no cumplen con los requisitos de calidad necesaria para ser utilizados.

En base a las especificaciones, se decidió considerar como aceptables aquellas mediciones con un valor de Usefulness asociado comprendido entre la mínima(1100) y máxima(0000) calidad. Los pixeles con valores 1101, 1110 y 1111 no poseen la calidad mínima suficiente para ser utilizados.

Los pixeles de la imagen de NDVI donde se detecto calidad insuficiente fueron reemplazados por la media aritmética de los dos valores temporales adyacentes.

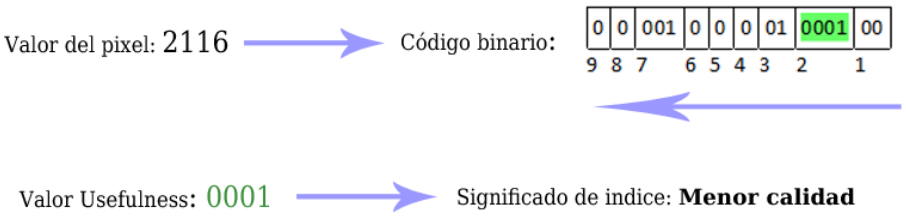


Figura 3.1: Ejemplo de conversión del producto Quality Assessment a código binario de 16 Bits. Las líneas verticales indican la separación entre los campos de información que contiene el código. En verde se resaltan los Bits correspondientes al índice Usefulnes.

3.1.2. Detección y corrección de valores atípicos utilizando una curva de tendencia

El método propuesto por Chen en "A simple method for reconstructing a high-quality NDVI time-series data set based on the Savitzky-Golay filter"[25] se basa en dos supuestos:

1. Los valores de los índices de vegetación tales como NDVI o EVI están fuertemente correlacionados al estado de la vegetación, por lo tanto la serie de tiempo de índices de vegetación tiende a seguir el patrón estacional de la vegetación a lo largo del año.

2. Las condiciones atmosféricas desfavorables para la observación de la superficie del planeta, generan una disminución en los valores obtenidos por el satélite.

En base a estos dos supuestos el autor propuso generar una serie temporal de tendencia "envolvente superior" a partir del filtro polinómico de Savitzky-Golay (el cual detallaremos a fondo más adelante). Según los resultados obtenidos por Chen se implementó un filtro con una ventana móvil de longitud nueve, un polinomio de sexto grado y una derivada de orden cero.

En nuestro caso hemos implementado simplificaciones en el método, haciendo el algoritmo más liviano computacionalmente.

El método se implemento de la siguiente manera:

1. Se extrajo la serie de tiempo.
2. Se aplicó el filtro de Savitzky-Golay, siendo la serie filtrada nuestra curva de tendencia.
3. Se creó una nueva serie de tiempo donde todos los valores de la serie original que estén por debajo de la curva de tendencia son remplazados por el valor de la curva de tendencia en ese punto.

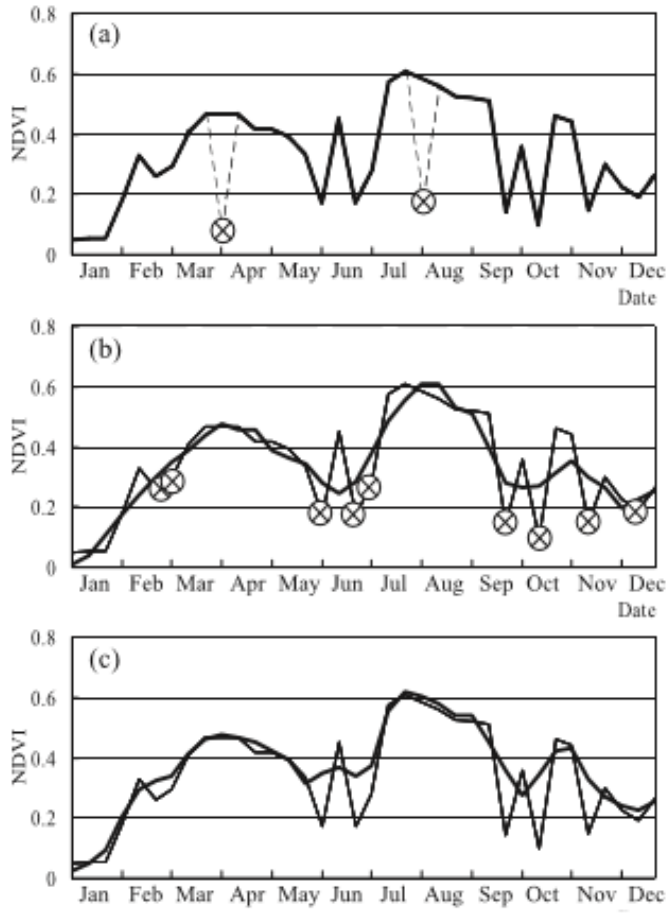


Figura 3.2: Ejemplo del aplicación de los métodos de extracción de valores anómalos. (a) Los datos anómalos detectados por la mascara de calidad son interpolados. (b) Se calcula la curva de tendencia(linea gruesa); los datos por debajo de esta son considerados anómalos. (c)Resultado de eliminar los datos anómalos (linea gruesa).[25]

3.2. Filtrado de series de tiempo

Si bien en la sección 3.1 hemos dados métodos para eliminar datos anómalos, las series de tiempo necesitan un mayor nivel de procesamiento para poder extraer de ellas características con precisión, por lo tanto introduciremos filtros que nos permitirán suavizar y ver tendencias las mismas.

3.2.1. Filtro Savitzky-Golay

Una forma de filtrar una serie de tiempo es remplazar cada valor de la serie con un nuevo valor obtenido de realizar una aproximación polinomial con $2n+1$ puntos vecinos (incluyendo el punto a filtrar) donde n es mayor o igual al grado del polinomio.

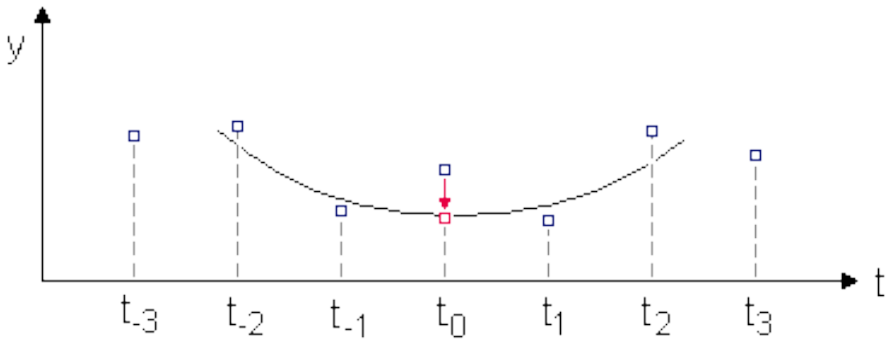


Figura 3.3: Aproximacion polinomial de grado 2 con 5 puntos[4].

Savitzky y Golay (1964) propusieron un ajuste polinomial móvil[35], que puede ser entendido como un promedio móvil ponderado, en donde la ponderación esta dada por un polinomio de cierto grado, dado que los coeficientes para todo el procedimiento son contantes para todos los valores de y[4].

Los coeficientes de dicho polinomio cuando son aplicados a una serie, realizan un ajuste por mínimos cuadrados dentro de la ventana del filtro.

El filtro puede ser aplicado a cualquier conjunto de datos consecutivos cuando los puntos son fijos e uniformes, y las curvas formadas graficando dichos puntos son continuas y más o menos lisas.Las series de NDVI satisfacen estas condiciones por lo que en filtro puede ser aplicado en ellas.

A continuación presentaremos la fórmula del filtro:

$$Y_j^* = \frac{\sum_{i=-m}^{i=m} C_i Y_{j+i}}{N} \tag{3.1}$$

Donde Y_j es el valor de NDVI original, Y_j^* es el NDVI resultante, C_i es el coeficiente para el i-ésimo valor de NDVI del filtro, N es el tamaño de la ventana, dado por $(2m+1)$. El indice j corre sobre la ordenada de la serie original, el

tamaño del filtro consiste en 2m+1 puntos, donde m el valor de la mitad de la ventana.

Los coeficientes pueden ser obtenidos directamente de Steinier et al. (1972) o calculados de las ecuaciones presentadas por Madden (1978).

Para aplicar el filtro a series de tiempo de NDVI hay dos parámetros que deben ser obtenidos empíricamente, el primero es m, usualmente un mayor valor de m produce más alisado pero achata los picos.

El segundo parámetro es el grado del polinomio (llamado "d") en general se utiliza un valor entre dos y cuatro.

Un grado pequeño producirá mas suavizado en la curva pero puede introducir sesgo, mientras que un valor grande puede reducir el sesgo pero sobre ajustar los datos y dar un resultado más ruidoso.[25]

3.2.2. Filtro Doble Logístico

Dada una serie de tiempo, la misma puede ser separada en ciclos fenológicos, donde cada ciclo esta compuesto por los datos pertenecientes a un lapso de tiempo contemplado entre dos mínimos locales.

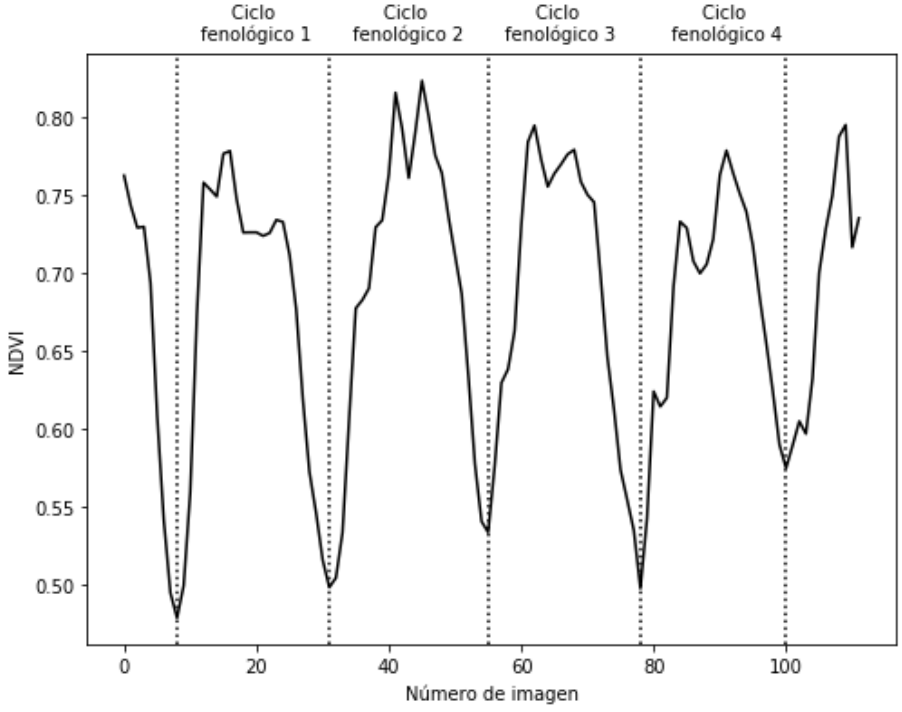


Figura 3.4: División de la serie de tiempo de NDVI en ciclos fenológicos.

El filtro doble logístico consiste entonces en ajustar una función que deno-

taremos como $F(t)$ a cada ciclo fenológico, $F(t)$ es la unión de tres funciones Doble logísticas locales (ver figura 3.6) contenidas en el ciclo fenológico.

La función local tiene la siguiente forma:

$$f(t) \equiv f(t; c, x) = c_1 + c_2 g(t : x) \tag{3.2}$$

donde:

$$g(t; x_1 ... x_4) = \frac{1}{1 + exp(\frac{x_1 - t}{x_2})} - \frac{1}{1 + exp(\frac{x_3 - t}{x_4})} \tag{3.3}$$

Los parámetros lineales $c = (c_1, c_2)$ están determinados por el nivel de la base y la amplitud de la curva.

Los parámetros no lineales $x = (x_1, x_2, x_3, x_4)$ determinan la forma de la función base $g(t; x)$, x_1 determinará la posición del punto de inflexión izquierdo, mientras que x_2 determinará la tasa de cambio, similarmente x_3 determinará la posición del punto de inflexión derecho y x_4 la tasa de cambio en ese punto.

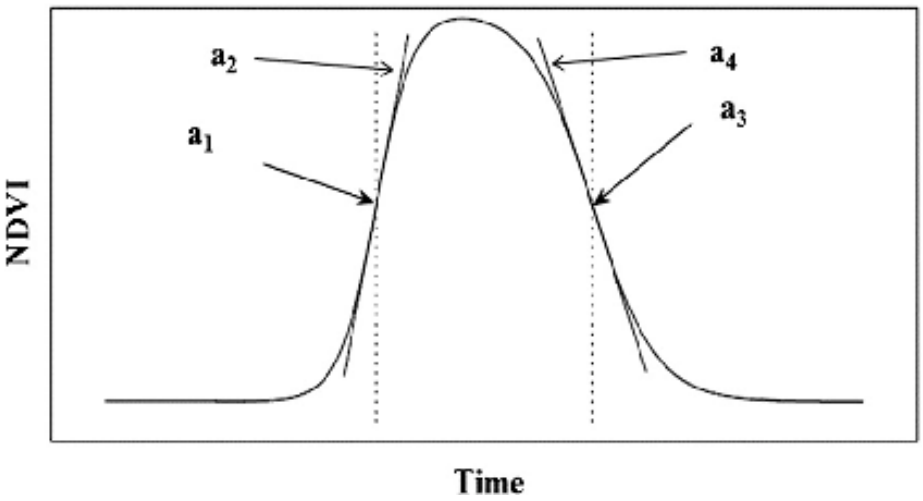


Figura 3.5: Función Doble logística, donde a1 es el punto de inflexión izquierdo y a2 es la tasa de cambio en ese punto; Similarmente a3 es el punto de inflexión derecho y a4 es la tasa de cambio es ese punto.[23]

Dado un conjunto de datos (t_i, y_i) i $= n_1, \ldots, n_2$ en un intervalo alrededor de un máximo o un mínimo, los parámetros c y x pueden ser estimados minimizando la función:

$$\chi^2 = \sum_{i=n_1}^{n_2} [f(t_i; c, x) - y_i]^2 \tag{3.4}$$

Estas funciones locales modelan muy bien la curva cerca del un mínimo o máximo, pero el ajuste va empeorando a medida que nos alejamos de ellos.

Denotamos las funciones locales que describen la serie de tiempo en intervalos alrededor del mínimo izquierdo, el máximo central y el mínimo derecho como f_L(t) , f_C(t) y f_R(t), la función global F(t) que describe la curva entre [t_L,t_R] es:

$$F(t) = \begin{cases} \alpha(t) f_L(t) + [1 - \alpha(t)] f_C(t) & t_L < t < t_C \\ \beta(t) f_C(t) + [1 - \beta(t)] f_R(t) & t_C < t < t_R \end{cases}$$

Donde $\alpha(t)$ y $\beta(t)$ son funciones que en un pequeño intervalo alrededor de $(t_L + t_C)/2$ y $(t_C + t_R)/2$ respectivamente, pasan suavemente de 1 a 0.

Unir las funciones locales el la clave del método,ya que incrementa la flexibilidad y permite un mayor ajuste de la función para acompañar el comportamiento complejo de la serie de tiempo.

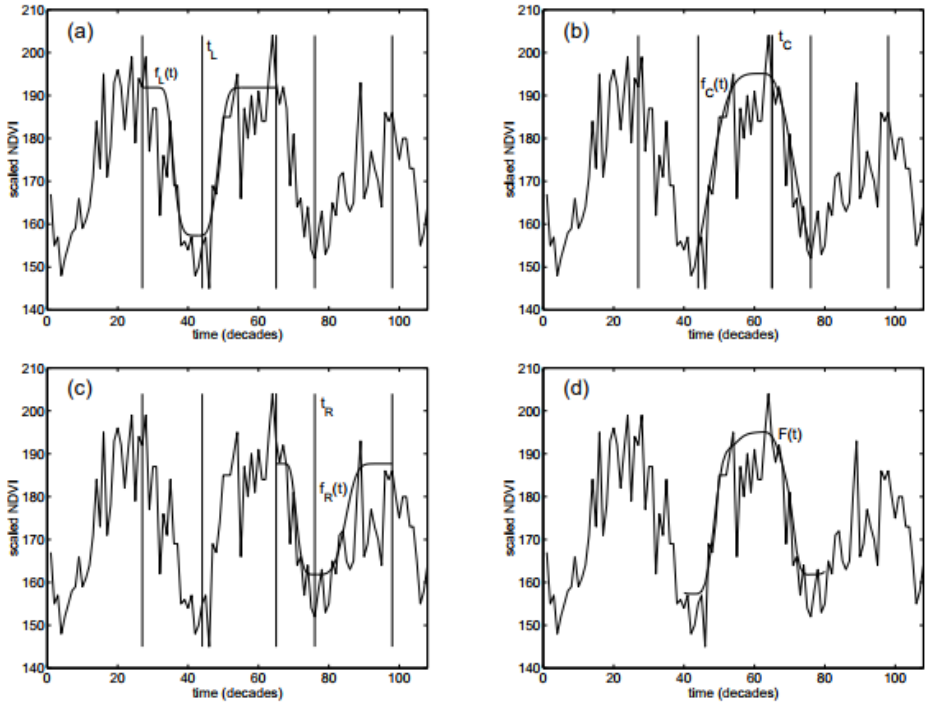


Figura 3.6: (a-c)Muestran el modelado de las funciones locales, (d)Muestra el modelo global obtenido al unir las funciones locales.[25]

Capítulo 4

Plugins de preprocesamiento

En este capítulo se describirán los plugins desarrollados para el pre procesamiento de las imágenes. Como se menciono anteriormente los plugins funcionan dentro del sistema de información geográfica QGIS, y están desarrollados en el lenguaje Python.

Hemos creado dos programas para el preprocesamiento de imágenes:

1. Plugin de eliminación de valores anómalos.
2. Plugin de filtrado de series de tiempo.

En las siguientes secciones detallaremos el funcionamiento de ambos.

4.1. Plugin de eliminación de datos anómalos

Este plugin implementa los métodos descritos en el capitulo anterior en la sección 3.1

4.1.1. Estructura

Para la creación de todos los plugins se utilizó un plugin provisto por QGIS llamado "Plugin Builder"[17], el cual crea un esqueleto del programa y genera los archivos necesarios para comenzar a trabajar.

A continuación se listan los archivos más importantes del plugin:

- **__init__.py, plugin_upload.py, resources.py**: Utilizados para cargar el plugin dentro del sistema QGIS.
- **outliers_eliminator.py**: Archivo principal del plugin donde se encuentra el algoritmo que elimina outliers ya sea utilizando una imagen de calidad y/o el método de curva de tendencia.

- **time_series.py**: Modela el tipo de dato "Serie de tiempo NDVI" contiene los métodos para interpolar y ajustar a la curva de tendencia.
- **outliers_eliminator_dialog.py, outliers_eliminator_dialog_base.ui**: Utilizados para crear la interfaz de usuario.
- **utils.py**: Contiene funciones auxiliares.

En cuanto a las bibliotecas de programacion (APIS) utilizadas podemos nombrar:

- **PyQt4**[15]: PyQt es un binding de la biblioteca gráfica Qt, utilizada para crear la interfaz de usuario.
- **GDAL**[5]: Biblioteca de software para la lectura y escritura de formatos de datos geoespaciales, utilizada en este caso para extraer información de capas raster existentes y crear nuevas, además nos permite iteractuar con QGIS.
- **SciPy, NumPy**[18][13]: Paquetes de computación científica para Python, SciPy contiene módulos para optimizar, álgebra lineal, integración, interpolación, funciones especiales, procesamiento de señales y de imagen y otras tareas para la ciencia e ingeniería, en tanto NumPy es una librería que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

4.1.2. Interfaz e inputs de la aplicación

Describiremos la interfaz de la aplicación, esto servirá como guía sobre el uso del plugin, ya que además se detallará como deben ser las entradas para el correcto funcionamiento del programa.

Figura 4.1: Interfaz del plugin

A continuación se detalla cada una de los elementos enumerados en la figura anterior (4.1):

1. Lista de capas raster abiertas en QGIS, la capa de entrada es un "stack"de

imagenes, es decir un conjunto de imagenes de NDVI ordenadas temporalmente alojadas cada una en un banda de la capa raster, se debe seleccionar la capa a la cual se le quiere eliminar datos anómalos, la misma debe tener por lo menos 23 bandas.

2. Dirección y nombre con el que se guardará la imagen resultante del proceso, es obligatorio.

3. Indica que se desea utilizar el método por curvas de tendencia para eliminar datos anómalos.

4. Número de iteraciones a correr el método por curva de tendencia (Sav-Gol).

5. Indica que se quiere utilizar máscara de calidad para eliminar outliers.

6. De indicarse que se desea utilizar una máscara de calidad, debe seleccionarse la misma de la lista de capas rasters disponibles, esta máscara debe tener las mismas dimensiones que la capa raster de entrada, y estar compuesta por las imagenes de calidad correspondientes a las imagenes de NDVI del primer item.

7. Dirección y nombre con el que se guardará la máscara de calidad a calcular, es opcional, si no se desea guardar la misma, puede dejarse en blanco.

8. Dirección y nombre con el que se guardará la máscara de resumen de calidad, es opcional, si no se desea guardar o calcular la misma, puede dejarse en blanco.

4.1.3. Funcionamiento

A continuación se describiremos el algoritmo principal de manera simplificada en forma de pseudocódigo.

Algoritmo 1 Eliminación de datos anómalos

```
1:  for i in rows do
2:    for j in cols do
3:      time_serie ← ndvi_raster[i][j]
4:      if use_quality then
5:        useful_serie ← is_useful(quality_raster[i][j])
6:        for n in useful_serie do
7:          if n is not useful then
8:            interpolate(time_serie, n)
9:          end if
10:       end for
11:     end if
12:     if use_savgol then
13:       for k = 0 to iter_number do
14:         savgol_serie ← savgol_filter(time_serie, 9, 6)
15:         for m = 0 to the length of time_serie do
16:           if time_serie[m] < savgol_serie[m] then
17:             time_serie[m] ← savgol_serie[m]
18:           end if
19:         end for
20:       end for
21:     end if
22:     result_raster[i][j] ← time_serie
23:   end for
24: end for
```

Donde:

- **ndvi_raster**: stack de imágenes, ndvi_raster[i][j] devuelve entonces un arreglo de todos los valores del pixel (i,j), a través del tiempo.
- **rows, cols**: Indica la cantidad de filas y columnas correspondientes a la imagen.
- **use_quality**: Bandera que determina si se debe o no utilizar el método de eliminación de outliers por mascara de calidad.
- **quality_raster**: stack de calidad.
- **use_savgol**: Bandera que determina si se debe o no utilizar le método de eliminación de outliers por curva de tendencia.

En cuanto a los métodos auxiliares mencionados en el pseudocódigo tenemos:

- **is_useful**(quality_serie): Toma una serie de valores de calidad (Sección 3.1.1), devuelve una nueva series donde, para cada elemento de la serie de calidad, convierte el elemento en un valor binario de 16 bits, extrae los bits que corresponden al valor de usefulness (bits del 10 al 14); si el valor indica que la calidad es aceptable ese elemento vale 1 en la nueva serie. Si no 0.
- **interpolate**(time_serie, n): Interpola el n-ésimo elemento de la serie con el promedio de sus vecinos.
- **savgol_filter**(time_serie, v, p): Aplica un filtro de Savitzky-Golay a la serie con ventana v y orden p.

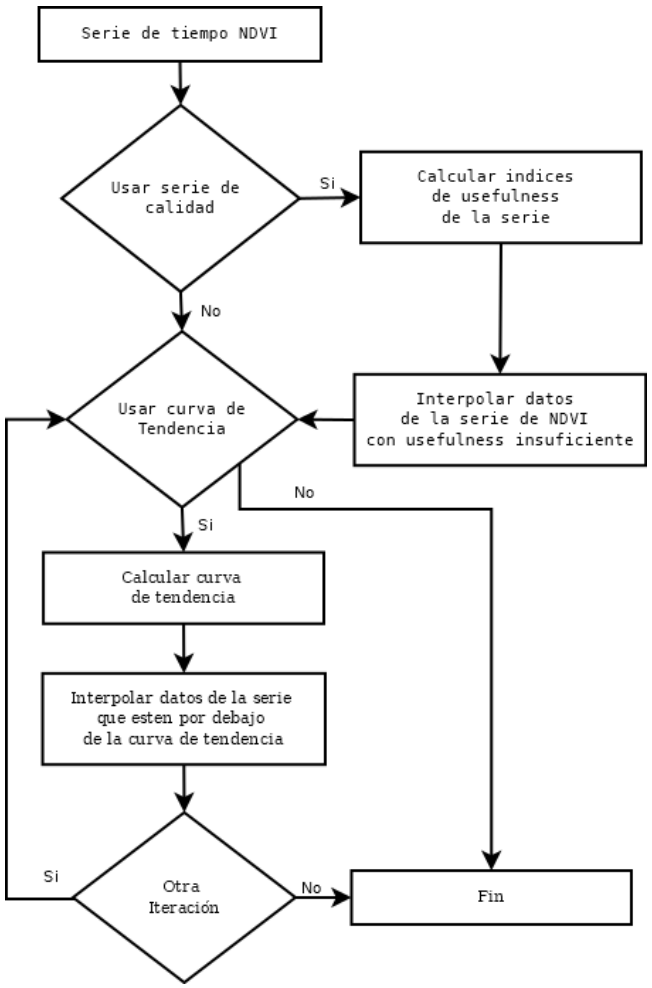


Figura 4.2: Diagrama de flujo del preprocesamiento de un pixel

En la siguiente figura se muestra los resultados de aplicar el algoritmo, con diferente número de iteraciones, a una serie de tiempo que corresponde a observar a lo largo del tiempo la posicion (i,j).

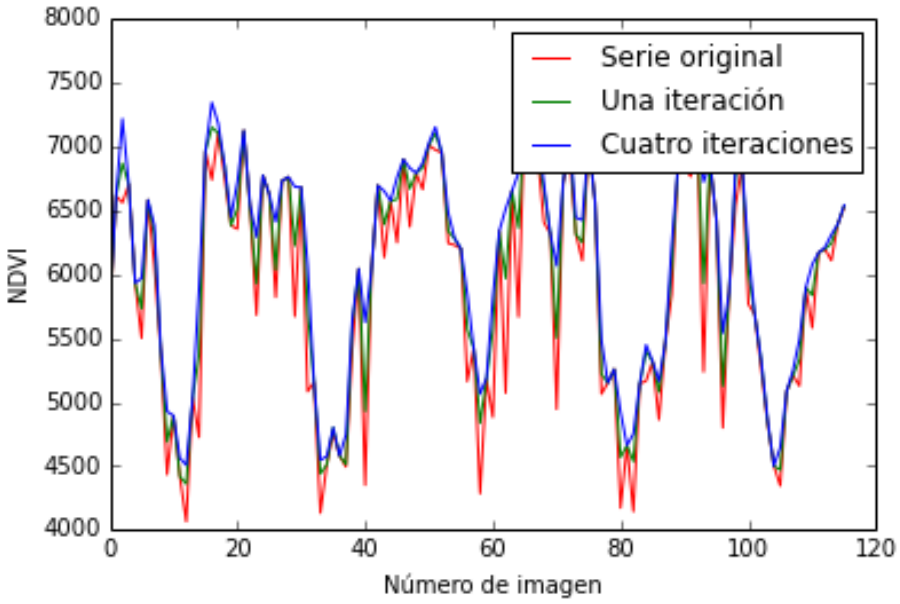


Figura 4.3: Comparación del resultado del método según el numero de iteraciones

Notar que aumentar el número de iteraciones ayuda a la mejor eliminación de datos anómalos cuando los mismos se manifiestan el picos muy pronunciados en la serie. Sin embargo esto tiene un limite y después de cierta cantidad de iteraciones el resultado se mantendrá constante. En nuestro caso hemos utilizado entre 3 y 5 iteraciones.

4.1.4. Outputs del plugin

Por último describiremos las salidas resultantes del plugin:

- **Stack Resultado**: Sera un stack de imágenes, idéntica a el stack NDVI de entrada pero con los datos que se han detectado anómalos interpolados.
- **Máscara de calidad**: Stack de igual dimensiones que el stack de calidad de entrada, donde el pixel (i,j,n) valdrá 1 si el valor de usefulness en el stack de calidad es aceptables y 0 en caso contrario.
- **Imagen de resumen**: Capa raster unibanda, donde el pixel (i,j) tiene el valor de cuantas veces falló la calidad para ese pixel en el tiempo.

4.2. Plugin de filtrado

En esta sección se describirá la implementación de el plugin que nos permite aplicar los filtros de Savitzky-Golay y Doble Logístico.

4.2.1. Estructura

Como se menciono en la sección 4.1.1 los plugins fueron creados con la misma herramienta ("Plugin Builder") por lo que la estructura del plugin es muy similar a la del plugin de eliminación de outliers.

A continuación de listan los archivos del plugin con una breve descripción:

- **__init__.py, plugin_upload.py, resources.py**: Utilizados para cargar el plugin dentro del sistema QGIS.
- **time_series_filters.py**: Archivo principal de la aplicación donde se describe parte del comportamiento de la interfaz y en el que se encuentra el algoritmo principal que aplica los distintos filtros al stack de entrada.
- **time_serie.py**: Modela una abstracción para las serie de tiempo, contiene la implementación de los filtros.
- **time_series_filter_dialog.py, time_series_filter_dialog_base.ui**: Utilizados para crear la interfaz de usuario.
- **utils.py**: Contiene funciones auxiliares.

Las APIS utilizadas son las mismas que en el plugin anterior por lo que se puede consultar en la sección 4.1.1 por las mismas.

4.2.2. Interfaz gráfica e Inputs de la aplicación

A modo de guía en el uso del plugin en esta sección describiremos la interfaz de usuario del plugin.

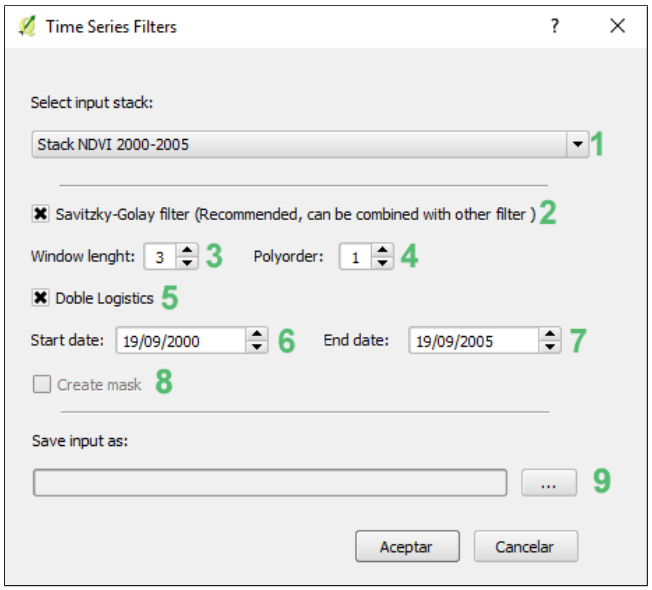


Figura 4.4: Interfaz del plugin

A continuación se detalla cada una de los elementos enumerados en la figura anterior (4.4):

1. Lista de capas raster abiertas en QGIS. Al menos una capa debe estar abierta para iniciar la aplicación, se debe seleccionar la capa que contiene las imágenes que se desean filtrar, la misma deben tener por lo menos 23 bandas, debe empezar y terminar en un máximo.

2. Indica que se quiere utilizar el filtro de Savitzky-Golay, este filtro puede usarse el combinación con el filtro Doble Logístico y en ese caso se aplicara primero que el filtro Doble Logístico.

3. Tamaño de la ventana, debe ser un numero natural impar.

4. Orden del polinomio, debe ser un numero natural menor al tamaño de la ventana. Es uno de los parámetros del filtro de Savitzky-Golay junto con el item anterior.

5. Indica que se quiere utilizar el filtro Doble Logístico, puede usarse en combinación con el filtro de Savitzky-Golay.

6. Fecha de inicio de las series de tiempo, corresponde a la fecha de la primera imagen del stack.

7. Fecha de finalización de las series de tiempo, corresponde a la fecha de la última imagen del stack. Junto con el item anterior se utiliza para determinar la cantidad de años a trabajar en el filtro Doble Logístico.

8. Indica si se desea crear una mascara, la cual informa que tan bien fue modelado cada año de una serie de tiempo.

9. Dirección y nombre con el que se guardará el stack de imagenes resultante del proceso.

4.2.3. Funcionamiento

Es esta sección describiremos los algoritmos desarrollados para la aplicación de los filtros.

Algoritmo 2 Filtrado de series temporales

```
1: for i in rows do
2:   for j in cols do
3:     time_serie ← ndvi_raster[i][j]
4:     if use_savgol then
5:       time_serie ← savgol(time_serie, win_length, order)
6:     else if use_doble_logistic then
7:       time_serie ← doble_logistic(time_serie)
8:     end if
9:     result_raster[i][j] ← time_serie
10:  end for
11: end for
```

Donde:

- **ndvi raster**: stack de imágenes, ndvi_raster[i][j] devuelve un arreglo de todos los valores del pixel (i,j), a través del tiempo.
- **rows, cols**: Cantidad de filas y columnas correspondientes a la imagen.
- **win_length, order**: parámetros de entrada del filtro Savizky_Golay.

En cuanto a la implementación de los filtros tenemos:

- **savgol**(time_serie, win_lenght, order):Llamada a la función savgol_filter de la libreria **scipy.signal**.
- **doble_logistico**(time_serie): Algoritmo que aplica la implementación del filtro Doble Logístico a la serie; a continuación lo describiremos con mas detalle.

Algoritmo 3 Detección de máximos y mínimos

1: *#Determinación de los mínimos anuales:*
2: **for** $i = 0$ to $years$ **do**
3: $minima[i] \leftarrow argmin(ndvi\_data[(i * 23)...((i + 1) * 23 + 1)]) + i * 23$
4: **end for**
5: *#Determinación de los máximos anuales:*
6: $maxima[0] \leftarrow argmax(ndvi\_data[0...minima[0]])$
7: **for** $i = 0$ to $years - 1$ **do**
8: $maxima[i + 1] \leftarrow argmax(ndvi\_data[minima[i]...minima[i + 1]]) + minima[i]$
9: **end for**
10: $maxima[years] \leftarrow argmax(ndvi_data[minima[years - 1]...]) + minima[years - 1]$

Donde:

- **years**: Cantidad de años completos que abarca la serie de tiempo.
- **ndvi_data**: Arreglo que representa la serie de tiempo de NDVI.
- **argmin, argmax**: Índice del mínimo, máximo correspondientemente.

El primer paso para aplicar el filtro es determinar los ciclos fenológicos de la serie temporal .Para ello le pedimos al usuario que la serie comience en un máximo (para facilitar la búsqueda del primer mínimo), luego tomamos segmentos de 23 datos (23 imágenes equivalen a un lapso de un año) y calculamos donde se encuentra el mínimo de cada segmento.Por último calculamos los máximos, en cada segmento formado por los mínimos.

Luego aplicamos el ajuste a la función logística tanto a las curvas que quedan delimitadas por mínimos (es decir alrededor de un máximo), como a las que quedan delimitadas por máximos(alrededor de un mínimo). (Ver figura 3.6)

Algoritmo 4 Aplicación del filtro Doble Logístico

1: *#Ajuste local para las curvas alrededor del mínimos:*
2: **for** $i = 0$ to $maxima\_size - 1$ **do**
3: $min\_fits[i] \leftarrow fit\_data(ndvi\_data[maxima[i]...[maxima[i+1]]])$
4: **end for**
5: *#Ajuste local para las curvas alrededor del máximos:*
6: **for** $i = 0$ to $minima\_size - 1$ **do**
7: $max\_fits[i] \leftarrow fit\_data(ndvi\_data[minima[i]...[minima[i+1]]])$
8: **end for**
9: *#Fusión de las curvas locales*
10: $result[...minima[0]] \leftarrow ndvi\_data[...minima[0]]$
11: **for** $i = 0$ to $maxima\_size - 2$ **do**
12: $min1 \leftarrow minima[i]$
13: $min2 \leftarrow minima[i+1]$
14: $max1 \leftarrow maxima[i]$
15: $max2 \leftarrow maxima[i+1]$
16: $result[min1...min2] \leftarrow fit\_marge(min\_fits[i], max\_fits[i], min\_fist[i+1], min1, min2, max1, max2)$
17: **end for**
18: $result[min2...] \leftarrow ndvi\_data[min2...]$

Donde:

- **máxima_size** : Tamaño del arreglo de máximos anuales de la series.
- **fit_data** y **fit_marge** se explicán a continuación.

En el siguiente algoritmo se muestra el algoritmo que realiza el ajuste de las curvas locales y la estimación inicial de los parámetros vistos en la sección 3.2.2.

Algoritmo 5 Función fit_data, ajuste de curva local

1: $c1 \leftarrow min(ndvi\_data)$ {c1: Nivel basal}
2: $c2 \leftarrow max(ndvi\_data) - c1$ {c2: Amplitud}
3: $x1, x3 \leftarrow inflection\_points(ndvi\_data)$ {x1, x3: Puntos de inflexión}
4: $x2 \leftarrow (ndvi\_data[x1] - ndvi\_data[0])/x1/100$ {x2: Tasa de cambio izquierda}
5: $last \leftarrow ndvi\_data\_size - 1$
6: $x4 \leftarrow (ndvi\_data[x3] - ndvi\_data[last])/(last - x3)/100$ {x2: Tasa de cambio derecha}
7: $c \leftarrow c1, c2$
8: $x \leftarrow x1, x2, x3, x4$
9: $estimate\_c, estimate\_x \leftarrow curve\_fit(f, t\_data, ndvi\_data, (c, x))$
10: $result \leftarrow f(data\_t, estimate\_c, estimate\_x)$

Donde:

- **min, max**: Funciones que devuelven el valor mínimo y máximo correspondientemente
- **curve_fit**: Función de la librería "scipy.optimize", usa un ajuste no linear por mínimos cuadrados para estimar los parámetros de una función f, dado una estimación inicial de los mismos.
- **f**: Función que modela la curva, descripta en el capitulo anterior.
- **inflection_points**: Función que devuelve una estimación de los dos puntos de inflexión de la curva para esto utilizamos un vector de diferencias y un vector con los signos de esas diferencias.

Por último se realiza la unión de curvas locales para lograr el modelado de un ciclo fenológico que compone la serie. Este proceso se repite para cada ciclo fenológico completo.

Sea:

- r_l_c = El lado derecho de la curva izquierda.
- l_c_c = El lado izquierdo de la curva central.
- r_c_c = El lado derecho de la curva central.
- l_r_c = El lado izquierdo de la curva derecha.

Algoritmo 6 Función fit_merge

1: $r\_l\_c \leftarrow left\_curve\_fit[min1 - max1...]$
2: $l\_c\_c \leftarrow center\_curve\_fit[...max2 - min1]$
3: $left\_result \leftarrow smooth\_vec * r\_l\_c + (1 - smooth\_vec) * l\_c\_c$
4: $r\_c\_c \leftarrow center\_curve\_fit[max2 - min1...]$
5: $l\_r\_c \leftarrow right\_curve\_fit[...min2 - max2]$
6: $right\_result \leftarrow smooth\_vec2 * r\_c\_c + (1 - smooth\_vec) * l\_r\_c$
7: $result \leftarrow concatenate(left_result, right_result)$

Donde:

- **smoth_vec, smoth_vec2**: son vectores del mismo tamaño que las curvas a unir, sus valores van progresivamente del 1 al 0. Son obtenidos por la funcion smooth_step.[19]

4.2.4. Outputs de la aplicación

El resultado de la aplicación será un nuevo stack de imágenes donde todas las series de tiempo han sido suavizadas y modeladas.

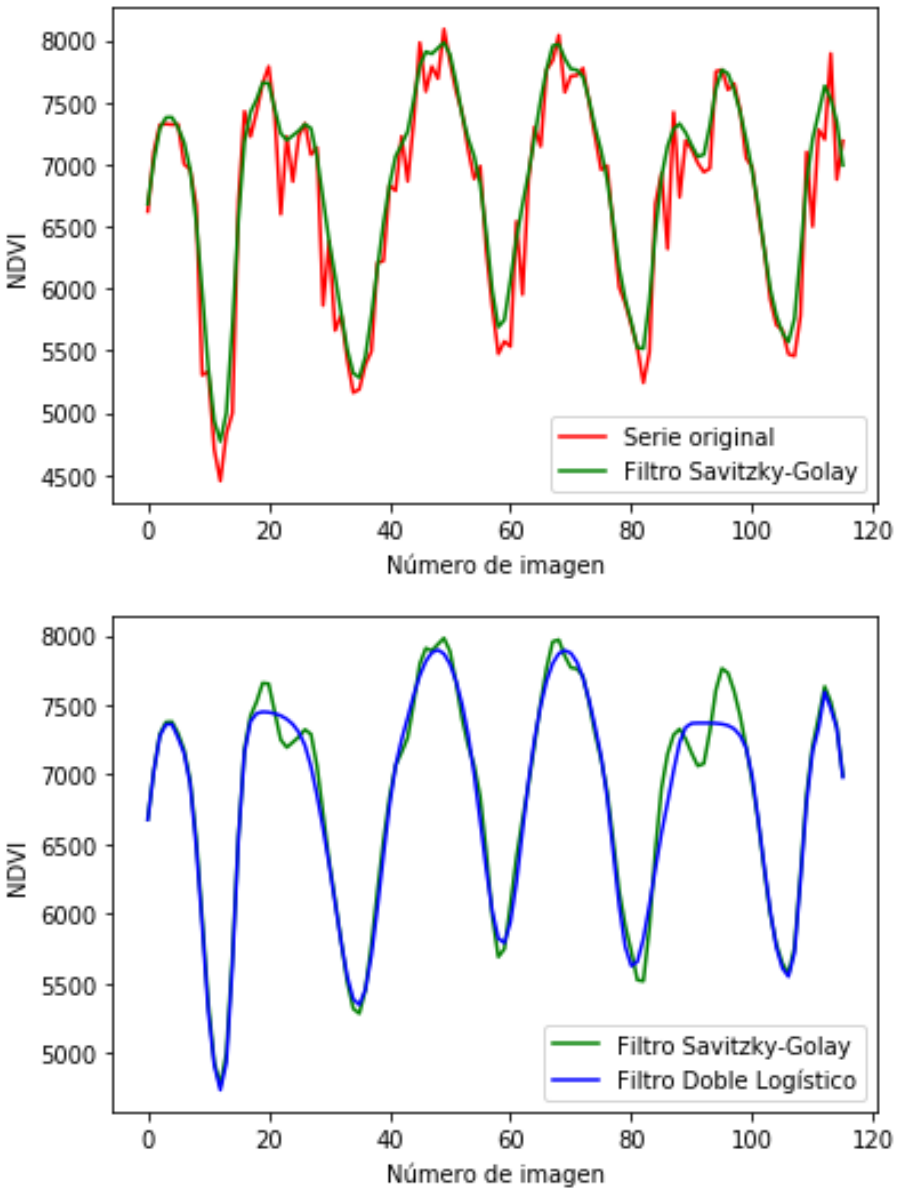


Figura 4.5: Resultado del filtrado de una serie de tiempo

Se recomienda utilizar primero el filtro se Savitzky-Golay para realizar un pre-suavizado de la serie y luego el filtro Doble Logístico, de esta manera se obtendrán mejores resultados en el ajuste de las series.

Capítulo 5

Extracción de Fenométricas

La fenología de las plantas estudia el ciclo vegetativo de las mismas a lo largo del año. Debido a que los indices de vegetación(IV) se hallan fuertemente relacionados con la cantidad de biomasa fotosintéticamente activa presente en el ecosistema, la serie temporal de IV refleja el comportamiento cíclico de la vegetación a lo largo del año.

[32]

Algunas de las fenologías que nos interesa extraer de nuestras series de tiempo de NDVI son:

- **I-NDVI**: Integral de todos los valores de IV de un periodo determinado. Puede estimarse para todo el ciclo anual o para periodos específicos. Esta relacionado con la producción de la vegetación.
- **Fecha de inicio y finalización de la estación de crecimiento**: Son las fechas estimadas para los de puntos de inflexión detectados en los modelos de vegetación. Biológicamente determinan el inicio y finalización del ciclo vegetativo de ganancia de carbono.
- **Fecha del pico de actividad fotosintética**: Fecha en que la vegetación alcanza el máximo valor de IV en el ciclo anual. Se corresponde con el momento del ciclo anual de máximo vigor productivo.
- **Valor del pico y del mínimo de la actividad fotosintética**: Máximo y mínimo valor de IV en el ciclo anual; tiene que ver con la producción anual de la vegetación.
- **Duración de la estación de crecimiento**: Número de días entre el inicio y finalización de la estación de crecimiento en los que la vegetación posee ganancia neta de carbono.
- **Amplitud y Rango relativo**: (Pico-Mínimo)/I-NDVI, biológicamente este cociente es una medida de la variación intra-anual de la actividad fotosintética.

- **Tasa de crecimiento o decrecimiento del IV**:Pendiente de la curva IV medida entre dos fechas; es la velocidad en el desarrollo de las fases de rebrote y senescencia.

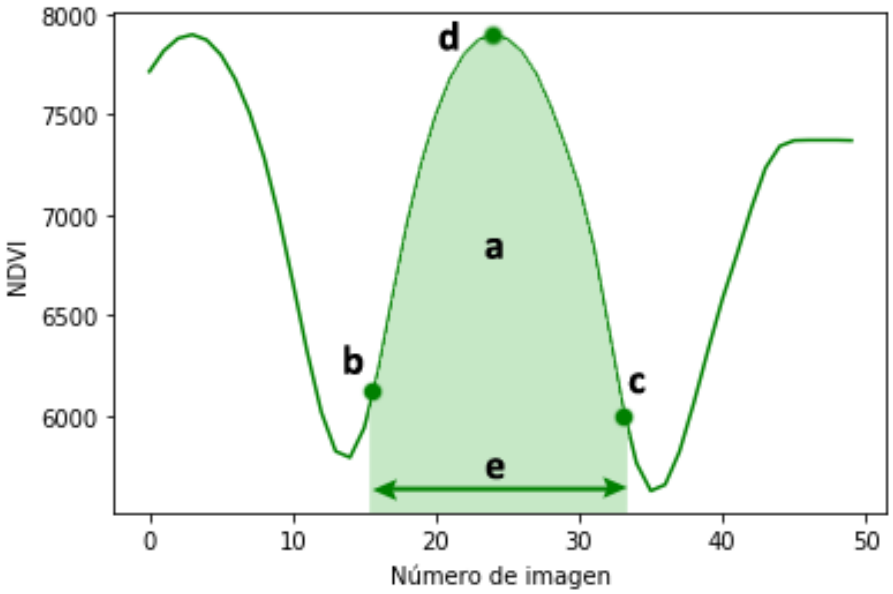


Figura 5.1: Algunas de las fenológias obtenidas: a) Integral b) Inicio de la estación de crecimiento c) Final de la estación de crecimiento b) Máximo e) Duración de la estación de crecimiento

A través de la obtención de características descriptivas de la serie temporal anual de IV es posible caracterizar el comportamiento de la vegetación a lo largo del año. Mediante esta técnica es posible detectar y cuantificar la fecha y la magnitud de los eventos relacionados a la captación de carbono y generación de biomasa.[32]

Estas métricas pueden utilizarse, por ejemplo, para generar distintos tipo de clasificaciones o aplicar métodos de muestreos. En el siguiente capitulo describiremos el plugin que extrae las fenométricas de las series de tiempo preprocesadas en el capitulo anterior.

Capítulo 6

Plugin de Extracción de Fenométricas

En las siguientes secciones veremos en detalle cuáles son y cómo se obtienen las fenoménicas incluidas en el plugin.

Dado que el programa fue creado con las mismas herramientas y utiliza las mismas bibliotecas que los plugins anteriores comparte gran parte de la estructura de los archivos que lo componen, por lo que no daremos muchos detalles sobre la misma, puede ser consultada en las secciones 4.1.1. y 4.2.1.

6.0.1. Inputs e Interfaz

En la siguiente imagen se muestra la interfaz de usuario de la aplicación y una breve descripción de sus componentes:

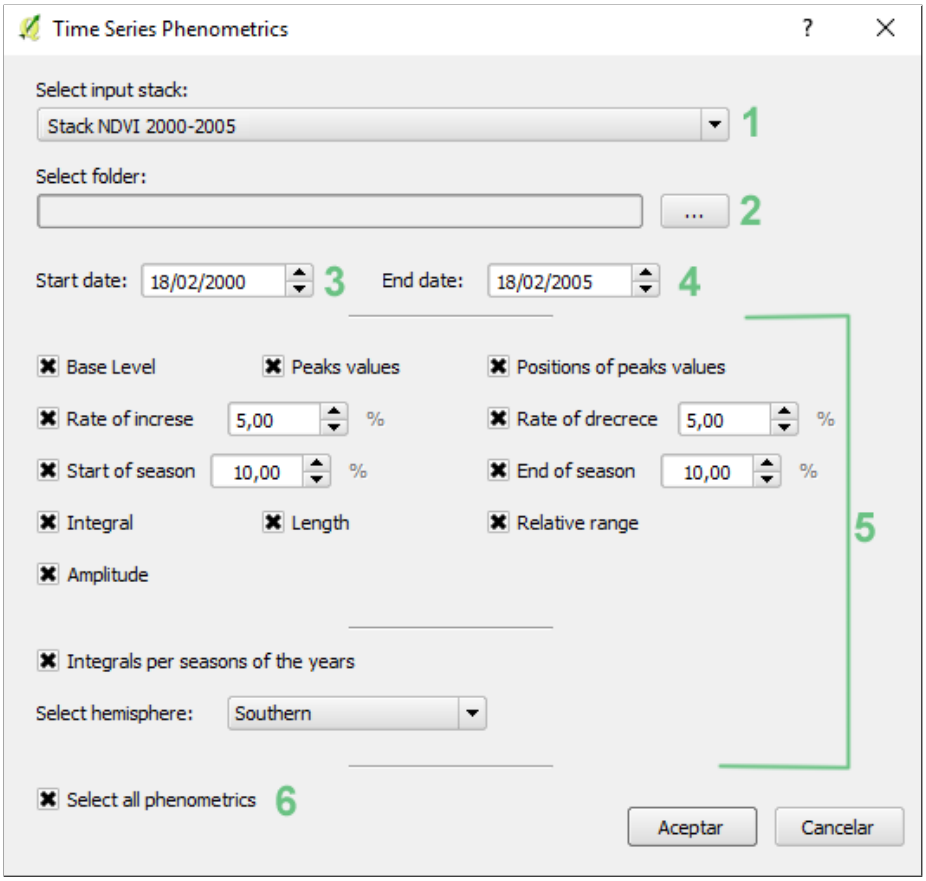


Figura 6.1: Interfaz del plugin

A continuación se detalla cada una de los elementos enumerados en la figura anterior (6.1):

1. Lista de capas raster abiertas en QGIS, al menos una capa debe estar abierta para iniciar la aplicación, se debe seleccionar la capa con la que se quiere trabajar, la misma debe tener por lo menos 46 bandas, debe empezar y terminar en un máximo.
2. Dirección de la carpeta donde se guardaran las imágenes resultado.
3. Fecha de la primera imagen.
4. Fecha de la última imagen.
5. Fenométricas a extraer
6. Atajo para seleccionar todas las fenométricas disponibles.

6.0.2. Algoritmos

En esta sección veremos en detalle el funcionamiento interno de la aplicación, empezaremos detallando el algoritmo principal del programa el cual nos deriva a las distintas subrutinas del plugin.

Algoritmo 7 Extracción de fenométricas

```
1:  for i in rows do
2:    for j in cols do
3:      time_serie ← ndvi_raster[i][j]
4:      if calculate_phenometric_0 then
5:        phenometric_0[i][j] ← get_phenometric_0(time_serie)
6:      end if
7:      .
8:      .
9:      .
10:     if calculate_phenometric_12 then
11:       phenometric_12[i][j] ← get_phenometric_12(time_serie)
12:     end if
13:   end for
14: end for
```

Donde:

- **ndvi_raster**: Stack de imágenes, ndvi_raster[i][j] devuelve entonces un arreglo de todos los valores del pixel (i,j), a través del tiempo.
- **rows, cols**: Cantidad de filas y columnas correspondientes a la imagen.
- **calculate_phenometric_n** : Bandera que indica si la n-esima fenométrica debe ser calculada, donde n = 0 ... 12.
- **phenometric_n**: Matriz donde se guardara el resultado de calcular la n-esia métrica.
- **get_phenometric_n**: Función que calcula la n-esima fenométrica para una serie dada.

A continuación se describe cada una de las doce características que se mencionaron en el algoritmo anterior.

Dada una serie de tiempo podemos extraer los mínimos de las misma utilizando el algoritmo de "detección de máximos y mínimos"de la sección 4.2.3, estos mínimos dividirán la serie de tiempo en distintos ciclos fenológicos, luego para cada ciclo de la serie obtendremos una característica.

Nivel basal (Basal level)

Para cada ciclo de la serie de tiempo el nivel basal estará dado por el promedio de los valores de NDVI de los mínimos que delimitan ese ciclo fenológico.

Posición de los máximos anuales (Position of peaks values)

Se calcula como vimos en el algoritmo de "detección de máximos y mínimos"de la seccion 4.2.3

Máximos anuales (Peak value)

Es el valor de NDVI en las posiciones obtenidas en el item anterior.

Tasa de crecimiento (Rate of increse)

Dado un porcentaje elegido por el usuario, para cada año, tomemos el lado izquierdo de la curva (desde el mínimo izquierdo hasta el máximo), sea (x1, y1) el primer punto tal que y1 es mayor que el ndvi del mínimo izquierdo en el porcentaje dado y sea (x2, y2) el último punto en el que y2 es inferior al ndvi del máximo en el porcentaje dado, luego la tasa estará dada por: $(y2-y1)/(x1-x2)$.

Tasa de decrecimiento (Rate of decrese)

Dado un porcentaje elegido por el usuario, para cada año, tomemos el lado derecho de la curva (desde el máximo hasta el mínimo derecho), sea (x1, y1) el primer punto tal que y1 es menor el ndvi de máximo en el porcentaje dado, sea (x2, y2) el último punto en el que y2 supera a el ndvi de minimo derecho en el porcentaje dado, luego la tasa estará dada por: $(y2 - y1)/(x1 - x2)$.

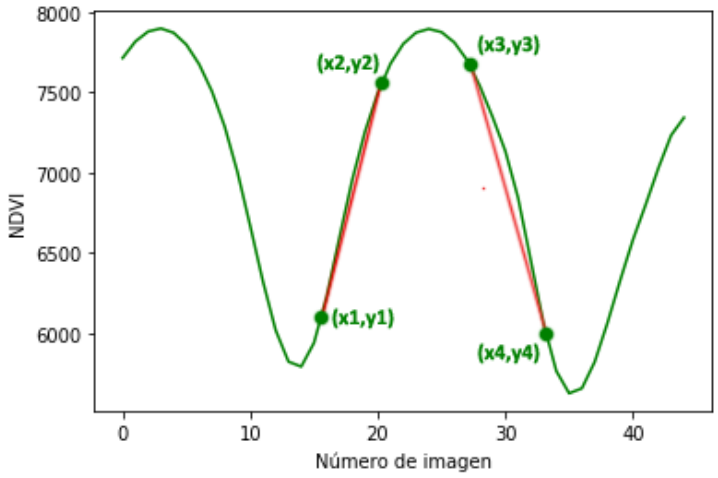


Figura 6.2: La pendiente de la recta que pasa por los puntos (x1,y1) e (x2,y2) será la tasa de crecimiento para ese ciclo fenológico. Similarmente, la pendiente de la recta formada por los puntos (x3,y3) e (x4,y4) será la tasa de decrecimiento para ese ciclo fenológico.

Inicio de la estación de crecimiento (Start of season)

Dado un porcentaje elegido por el usuario, para cada ciclo fenológico de la serie de tiempo, queremos encontrar el tiempo en el que el ndvi ha crecido en ese porcentaje con respecto al mínimo izquierdo.

Como nuestras series de tiempo son discretas, buscamos el primer punto del año donde el valor de NDVI es mayor a ese valor, luego estimamos una recta con el punto anterior, por último buscamos el punto deseado en esa recta.

Algoritmo 8 Extracción de inicio de la estación de crecimiento para una serie de tiempo.

1: **for** $y = 0$ to $years$ **do**
2: $min1 \leftarrow min\_t[y]$
3: $sos\_ndvi = time\_serie[min1] + time\_serie[min1] * perc$
4: $t = min1 + 1$
5: **while** $time\_serie[t] < sos\_ndvi$ **do**
6: $t = t + 1$
7: **end while**
8: $a \leftarrow time\_serie[t] - time\_serie[t-1]$
9: $b \leftarrow time\_serie[t] - a * t$
10: $result\_sos[i] \leftarrow (sos\_ndvi - b)/a$
11: **end for**

Donde:

- **time_serie**: Serie de tiempo extraída de la imagen.
- **years**: Año de la última imagen del stack - Año de la primera imagen del stack - 1.
- **min_t**: Mínimos de la serie de tiempo.
- **perc**: Porcentaje elegido por el usuario.
- **result_sos**: Arreglo donde se guardan los resultados.

Fin de la estación de crecimiento (End of season)

Similarmente al punto anterior, el fin de la estación de crecimiento dependerá de un porcentaje elegido por el usuario, en el lado derecho de la curva.

Algoritmo 9 Extracción de inicio de la estación de crecimiento para una serie de tiempo.

```
for y = 0 to years do
    min2 ← min_t[y + 1]
    eos_ndvi = time_serie[min2] + time_serie[min2] ∗ perc
    t = min2 − 1
    while time_serie[t] < eos_ndvi do
        t = t − 1
    end while
    a ← time_serie[t + 1] − time_serie[t]
    b ← time_serie[t] − a ∗ t
    result_sos[y] ← (eos_ndvi − b)/a
end for
```

Donde:

- **time_serie**: Serie de tiempo extraída de la imagen.
- **years**: Año de la última imagen del stack - Año de la primera imagen del stack - 1.
- **min_t**: Mínimos de la serie de tiempo.
- **perc**: Porcentaje elegido por el usuario.
- **result_eos**: Arreglo donde se guardan los resultados.

Integral(Integral)

Integral de la curva entre el inicio y el final de la estación de crecimiento, utilizamos la funcion "**trapz**"de la libreria numpy para python, la cual utiliza la regla del trapezoide para integrar.

Duración de la estación (Length)

Para cada ciclo fenológico la duración de la estación de crecimiento esta dada por la diferencia entre el final y comienzo de la estación de crecimiento obtenida anteriormente.

Amplitud (Amplitude)

La amplitud de cada ciclo fenológico estará dada por la diferencia entre el máximo y el nivel basal de ese ciclo.

Rango relativo (Relative range)

El rango relativo esta dado por la división entre la amplitud y la integral de cada ciclo fenológico.

Integral por estaciones del año(Integrals per seasons of the year)

En estas fenométrica para cada año obtenemos cuatro integrales, una por cada estación que divide el año. Para ello debemos encontrar que número de imagen corresponde al primer verano (para el hemisferio sur) o el primer invierno (para el hemisferio norte) de la serie; esto es fácil ya que el usuario nos da como entrada la fecha de la primera imagen, luego como sabemos que por año disponemos de 23 imágenes, podemos extraer los años de la serie moviéndonos de 23 en 23 imágenes a partir de la imagen del comienzo del primer verano.

A cada año de la serie lo dividimos de la siguiente manera:

- Desde el 19 de diciembre al 22 de marzo: Estación de **verano** en el hemisferio sur e invierno en el hemisferio norte.
- Desde el 22 de marzo al 26 de junio: Estación de **otoño** en el hemisferio sur y primavera en el hemisferio norte.
- Desde el 26 de junio al 30 de septiembre: Estación de **invierno** en el hemisferio sur y verano en el hemisferio norte.
- Desde el 30 de septiembre al 19 de diciembre: Estación de **primavera** en el hemisferio sur y otoño en el hemisferio norte.

Cuando identificamos qué imágenes corresponden a cada estación del año, integramos las mismas con la función **trapz** de numpy, mencionada anteriormente.

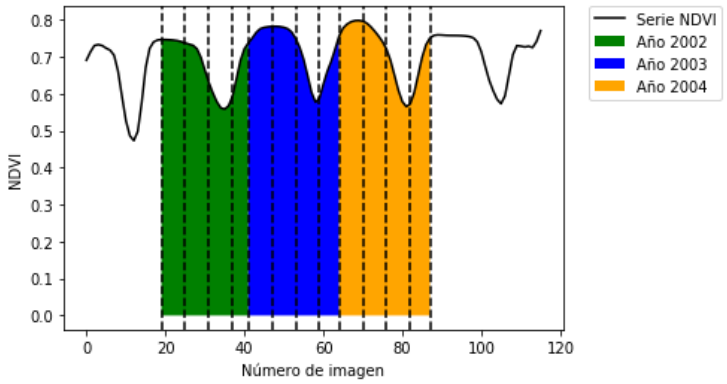


Figura 6.3: La figura muestra para cada año las integrales para sus estaciones.

6.0.3. Outputs

Como resultado del proceso de extracción de fenologías obtendremos distintos stack de imágenes, uno por cada parámetro solicitado, los mismos se guardaran en la carpeta que hayamos seleccionado en un principio.

A continuación se muestran, a modo de ejemplo, dos series de tiempo de NDVI de lugares distintos, y se compararam sus fenologías:

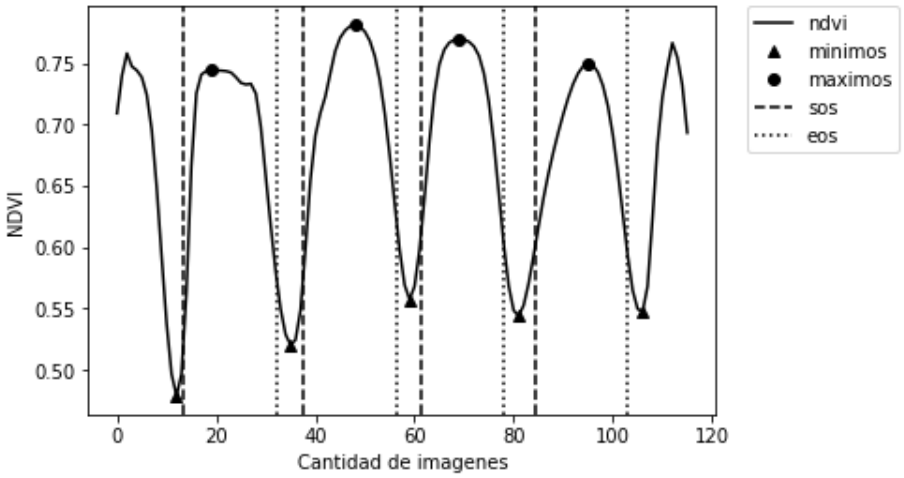


Figura 6.4: Serie de tiempo de NDVI del Bosque Chaqueño, obtenida entre 2000 y 2005

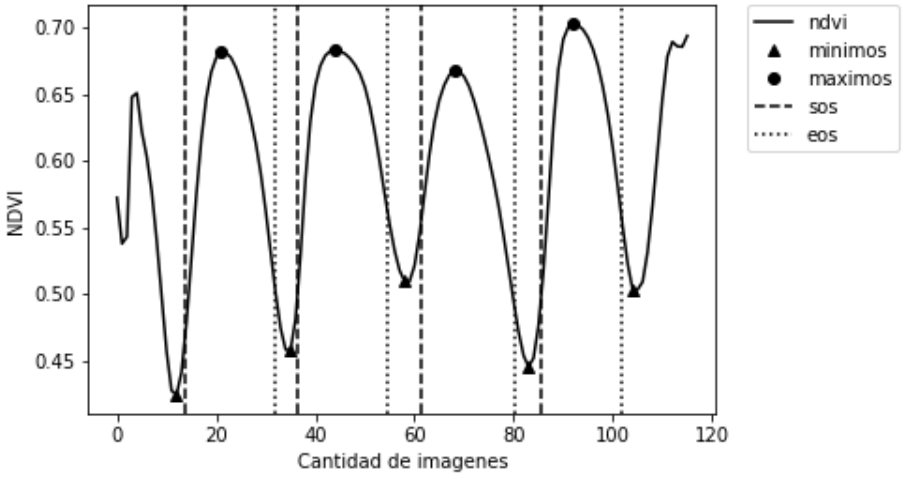


Figura 6.5: Serie de tiempo de NDVI de las Sierras de Córdoba, obtenida entre 2000 y 200

Cuadro 6.1: Comparación de máximos anuales

Máximos anuales		
	Bosque Chaqueño	Sierras de Córdoba
2001	0.7441	0.6379
2002	0.7811	0.587
2003	0.7697	0.6013
2004	0.7499	0.7122

Las distintas fenologías extraídas de las series de tiempo nos permiten realizar comparaciones entre series de distintos lugares, clasificaciones e implementaciones de métodos de muestreo, probando ser útiles para la investigación y toma de decisiones sobre el ecosistema.

Capítulo 7

Concluciones

7.1. Comparación con TIMESAT

Como hemos mencionado al principio de este trabajo (sección 2.5) TIMESAT es uno de los programas más prestigiosos para pre-procesar, modelar series de timpo y calcular fenología de las mismas. La idea de este trabajo era crear una batería de software que no solo cubriera las funcionalidades de TIMESAT si no también tratara algunas sus desventajas.

Uno de los principales obstáculos que presenta TIMESAT es que no se encuentra incorporado como herramienta en un sistema de información geográfica, por lo que puede llegar a complicarse su uso a nuevos usuarios. Este hecho también lo limita en cuanto recursos para la manipulación de imágenes en general, por lo que debe ser combinado con otras herramientas. En nuestro caso elegimos el sistema de información geográfica QGIS ya que provee un entorno completo para trabajar, además se puede agregar software de terceros a través de scrips y plugins que pueden descargarse facilmente.

Otro punto en contra de bien TIMESAT es que si bien es un programa de uso gratuito, no es de código abierto; en contraste nuestro programa es de código libre lo que permite que pueda ser modificado y mejorado por otros usuarios según sus necesidades.

Cabe aclarar que TIMESAT en un programa con muchos años de desarrollo y sus rutinas son estables y robustas, algunos de nuestros algoritmos tuvieron que ser simplificados y adaptados por lo que se pueden presentar más errores en nuestras rutinas.

TIMESAT automatiza la obtención de algunos parámetros, los cuales nosotros pedimos al usuario, si bien esto tiene como desventaja mayor trabajo para el usuario y variabilidad, simplifica significativamente el costo computacional.

7.2. Puntos a mejorar

A continuación se listan posibles cambios y agregados que fueron dejados para futuros desarrollos:

- En nuestro plugin de eliminación de valores anómalos utilizamos solo el campo *usefulnes* de las imágenes de calidad, por ser el indice más completo. Sin embargo estas imágenes tienen mas información como se ve en el cuadro 3.1, por lo que podría agregarse al programa la opción de personalizar la máscara de calidad según estos datos.
- En el plugin de filtros se puede implementar el filtro asimétrico gaussiano, que esta disponible en TIMESAT, pero no logramos implementarlo en el lenguaje Python.
- Por ultimo en nuestro plugin de extraccion de fenometricas alguna de las funciones pueden ser automatizadas y no necesitar de los inputs del usuario como es el caso del inicio y finalizacion de la estacion de crecimiento.

7.3. Conclusiones Finales

Dado los problemas que surgen en el momento de obtener imágenes satelitales, las mismas deben ser procesadas o adaptadas para obtener de ellas información útil para resolver problemas de la vida real. Si bien se cuenta con una gran cantidad de sistemas de información geográfica y otros recursos para el procesamiento de imágenes satelitales, aun falta mucho por desarrollar, sobre todo para las distintas tareas especificas en los diferentes campos que utilizan dichas imágenes.

Para la construcción de estos programas no solo se necesita conocimiento en desarrollo de software sino también en otras áreas, en nuestro caso biología y matemática, lo que constituye un interesante desafío multidisciplinario.

En nuestro trabajo desarrollamos software como aporte para el sistema de información geográfica QGIS, lo cual nos permite integrarlo a un entorno con otras funciones necesarias para la manipulación de imágenes, esperando no solo que sea una buena alternativa a otros programas mas restrictivos si no también que en el futuro se puedan ampliar sus funcionalidades o modificaras según sea necesario.

Bibliografía

[1] Canada centre for remote sensing. http://sar.kangwon.ac.kr/etc/fundam/chapter1/chapter1_5_e.html. Accessed: 2018-02-20.

[2] Enhanced vegetation index wikipedia. https://en.wikipedia.org/w/index.php?title=Enhanced_vegetation_index&oldid=822335943. Accessed: 2018-02-20.

[3] Función de distribución de reflectancia bidireccional. https://es.wikipedia.org/wiki/Funci%C3%B3n_de_distribuci%C3%B3n_de_reflectancia_bidireccional. Accessed: 2018-02-20.

[4] Fundamentals of stadistics. http://www.statistics4u.com/fundstat_eng/cc_filter_savgolay.html. Accessed: 2018-02-20.

[5] Gdal - geospatial data abstraction library page. http://www.gdal.org/. Accessed: 2018-02-20.

[6] Instituto de tegnologias educativas. https://fjferrer.webs.ull.es/Apuntes3/Leccion02/11_efecto_del_ngulo_de_incidencia_de_los_rayos_solares_sobre_la_irradiancia.html. Accessed: 2018-02-20.

[7] Moderate resolution imaging spectroradiometer. https://modis.gsfc.nasa.gov/. Accessed: 2018-02-20.

[8] Moderate-resolution imaging spectroradiometer (modis) wikipedia. https://es.wikipedia.org/w/index.php?title=Espectrorradi%C3%B3metro_de_im%C3%A1genes_de_media_resoluci%C3%B3n&oldid=102420468. Accessed: 2018-02-20.

[9] National centers for environmental information. https://www.ngdc.noaa.gov/. Accessed: 2018-02-20.

[10] Natural resources canada. http://www.nrcan.gc.ca/node/14635. Accessed: 2018-02-20.

[11] Índice de vegetación de diferencia normalizada nasa. https://earthobservatory.nasa.gov/Features/MeasuringVegetation/measuring_vegetation_2.php. Accessed: 2018-02-20.

[12] Índice de vegetación de diferencia normalizada wikipedia. https://es.wikipedia.org/w/index.php?title=%C3%8Dndice_de_vegetaci%C3%B3n_de_diferencia_normalizada&oldid=101961920. Accessed: 2018-02-20.

[13] Numpy library page. http://www.numpy.org/. Accessed: 2018-02-20.

[14] Phenex package documentation. https://www.rdocumentation.org/packages/phenex/versions/1.4-5. Accessed: 2018-02-20.

[15] Pyqt4 wiki page. https://wiki.python.org/moin/PyQt4. Accessed: 2018-02-20.

[16] Qgis. https://www.qgis.org/es/site/. Accessed: 2018-02-20.

[17] Qgis plugin builder page. https://plugins.qgis.org/plugins/pluginbuilder/. Accessed: 2018-02-20.

[18] Scipy library page. https://www.scipy.org/. Accessed: 2018-02-20.

[19] Smoothstep wikipedia. https://en.wikipedia.org/wiki/Smoothstep. Accessed: 2018-02-20.

[20] Timesat page. http://web.nateko.lu.se/timesat/timesat.asp. Accessed: 2018-02-20.

[21] Topografia, cartografia y ciencias afines. http://lanero.net/ViejaWeb/?in_id=sensores. Accessed: 2018-02-20.

[22] D. Alcaraz-Segura, C. Di Bella, and J. Straschnoy. *Earth Observation of Ecosystem Services*. 01 2013.

[23] E. Borges, E. Sano, and E. Silva. Radiometric quality and performance of timesat for smoothing moderate resolution imaging spectroradiometer enhanced vegetation index time series from western bahia state, brazil. 8:083580, 07 2014.

[24] J. Cabello, D. Alcaraz-Segura, and P. Lourenço. Funcionamiento de los ecosistemas de la red de parques nacionales de españa: detección de impactos recientes y desarrollo de un sistema de seguimiento y alerta a partir de herramientas de teledetección, 01 2012.

[25] J. Chen, P. Jönsson, M. Tamura, Z. Gu, B. Matsushita, and L. Eklundh. A simple method for reconstructing a high-quality ndvi time-series data set based on the savitzky-golay filter. 91:332–344, 06 2004.

[26] E. Chuvieco and E. Salinero. *Fundamentos de teledetección espacial*. Rialp, 1995.

[27] L. Eklundh and P. Jönsson. TIMESAT 3.2 with parallel processing-Software Manual. *Lund University*, 2012.

[28] X. Gao, A. R. Huete, W. Ni, and T. Miura. Optical–biophysical relationships of vegetation spectra without background contamination. *Remote Sensing of Environment*, 74(3):609 – 620, 2000.

[29] H. Kobayashi and D. Dye. Atmospheric conditions for monitoring the long-term vegetation dynamics in the amazon using normalized difference vegetation index. 97:519–525, 09 2005.

[30] C. Kuenzer, S. Dech, and W. Wagner, editors. *Remote Sensing Time Series*, volume 22. Springer International Publishing, 2015. DOI: 10.1007/978-3-319-15967-6.

[31] J. M. Paruelo. La caracterización funcional de ecosistemas mediante sensores remotos. *Revista Ecosistemas*, 17(3), 2008.

[32] N. Pettorelli, J. O. Vik, A. Mysterud, J.-M. Gaillard, C. J. Tucker, and N. C. Stenseth. Using the satellite-derived NDVI to assess ecological responses to environmental change. *Trends in Ecology & Evolution*, 20(9):503–510, Sept. 2005.

[33] D. P. Roy, J. S. Borak, S. Devadiga, R. E. Wolfe, M. Zheng, and J. Descloitres. The MODIS land product quality assessment approach. *Remote Sensing of Environment*, 83(1-2):62–76, 2002.

[34] E. Salinero. *Teledetección Ambiental: La Observación de la Tierra Desde el Espacio*. Ariel, 2002.

[35] A. Savitzky and M. Golay. Smoothing and differentiation of data by simplified least squares procedures. 36:1627–1639, 07 1964.

[36] J. Schott. *Remote Sensing: The Image Chain Approach*. Oxford University Press, 2007.

[37] R. Solano, K. Didan, A. Jacobson, and A. Huete. MODIS vegetation index user's guide (MOD13 series). *Vegetation Index and Phenology Lab, The University of Arizona*, pages 1–38, 2010.

Apéndice A

Código plugin de eliminación de datos anómalos

Principales archivos del plugin:

A.1. outliers_eliminator.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 OutliersEliminator
                                 A QGIS plugin
 Eliminates atypicals values of MODIS raster stacks
                              -------------------
        begin                : 2017-03-02
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
 ***************************************************************************/
"""
from PyQt4 import QtGui
from PyQt4 import QtCore
from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
```

```
from PyQt4.QtGui import (QAction, QIcon, QFileDialog)
from qgis.gui import QgsMessageBar


# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
from outiliers_eliminator_dialog import OutliersEliminatorDialog
import os.path
from time import time
import gdal
import numpy as np

from utils import (error_dialog, ProgressBar, string_to_raster)
from time_series import TimeSerie

class OutliersEliminator:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'OutliersEliminator_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Create the dialog (after translation) and keep reference
        self.dlg = OutliersEliminatorDialog()

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&MODIS_Time_Series_Outliers_Elimination')
        # TODO: We are going to let the user set this up in a future iteration
        self.toolbar = self.iface.addToolBar(u'OutliersEliminator')
        self.toolbar.setObjectName(u'OutliersEliminator')

        #Save as buttons funtionality
        self.dlg.lineEdit.clear()
        self.dlg.pushButton.clicked.connect(self.select_output_file)
        self.dlg.lineEdit_2.clear()
        self.dlg.pushButton_2.clicked.connect(self.select_output_file_2)
        self.dlg.lineEdit_3.clear()
        self.dlg.pushButton_3.clicked.connect(self.select_output_file_3)
        self.dlg.checkBox.stateChanged.connect(self.check_qua)
        self.dlg.checkBox_2.stateChanged.connect(self.check_sav_gol)

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.
```

```
    We implement this ourselves since we do not inherit QObject.

    :param message: String for translation.
    :type message: str, QString

    :returns: Translated version of message.
    :rtype: QString
    """
    # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
    return QCoreApplication.translate('OutliersEliminator', message)


def add_action(
    self,
    icon_path,
    text,
    callback,
    enabled_flag=True,
    add_to_menu=True,
    add_to_toolbar=True,
    status_tip=None,
    whats_this=None,
    parent=None):
    """Add a toolbar icon to the toolbar.

    :param icon_path: Path to the icon for this action. Can be a resource
        path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
    :type icon_path: str

    :param text: Text that should be shown in menu items for this action.
    :type text: str

    :param callback: Function to be called when the action is triggered.
    :type callback: function

    :param enabled_flag: A flag indicating if the action should be enabled
        by default. Defaults to True.
    :type enabled_flag: bool

    :param add_to_menu: Flag indicating whether the action should also
        be added to the menu. Defaults to True.
    :type add_to_menu: bool

    :param add_to_toolbar: Flag indicating whether the action should also
        be added to the toolbar. Defaults to True.
    :type add_to_toolbar: bool

    :param status_tip: Optional text to show in a popup when mouse pointer
        hovers over the action.
    :type status_tip: str

    :param parent: Parent widget for the new action. Defaults None.
    :type parent: QWidget

    :param whats_this: Optional text to show in the status bar when the
        mouse pointer hovers over the action.

    :returns: The action that was created. Note that the action is also
        added to self.actions list.
    :rtype: QAction
    """
    icon = QIcon(icon_path)
    action = QAction(icon, text, parent)
    action.triggered.connect(callback)
    action.setEnabled(enabled_flag)

    if status_tip is not None:
```

```
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToRasterMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/OutliersEliminator/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'MODIS_Time_Series_Outliers_Elimanation'),
            callback=self.run,
            parent=self.iface.mainWindow())


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginRasterMenu(
                self.tr(u'&MODIS_Time_Series_Outliers_Elimination'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    def clear_dialog(self):
        """
        Clean all el element on the main dialog
        """
        self.dlg.comboBox.clear()
        self.dlg.spinBox.setValue(1)
        self.dlg.lineEdit.clear()
        self.dlg.comboBox_2.clear()
        self.dlg.lineEdit_2.clear()
        self.dlg.lineEdit_3.clear()
        self.dlg.checkBox.setChecked(False)
        self.dlg.checkBox_2.setChecked(False)


    def get_layers(self):
        """
        Reads the layer to be load in the combos boxs.
        """
        layers = self.iface.legendInterface().layers()
        layer_list = []
        for layer in layers:
            if layer.type() == 1: #Layer is raster
                layer_list.append(layer.name())
        self.dlg.layers = layers
        return layer_list

    def fill_combo(self, combo_box, layer_list, is_optional):
        """Fill a combo_box with all the rasters layers

        :param combo_box: Combo to fill
```

```
        :type icon_path: QtComboBox

        :param is_optional: Indicates the combo has the first option as None
        :type text: bool
        """
        if is_optional:
            combo_box.addItems([None])
        combo_box.addItems(layer_list)

    def check_qua(self, state):
        is_enable = False
        if state == QtCore.Qt.Checked:
            is_enable = True
        self.dlg.comboBox_2.setEnabled(is_enable)
        self.dlg.pushButton_2.setEnabled(is_enable)
        self.dlg.pushButton_3.setEnabled(is_enable)

    def check_sav_gol(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.spinBox.setEnabled(True)
        else:
            self.dlg.spinBox.setEnabled(False)

    def select_output_file(self):
        """Connected to de first push button.
        """
        filename = QFileDialog.getSaveFileName(self.dlg,
        "Select output file ", "", '*.tiff')
        self.dlg.lineEdit.setText(filename)

    def select_output_file_2(self):
        """Connected to de seconf push button.
        """
        filename = QFileDialog.getSaveFileName(self.dlg,
        "Select output file ", "", '*.tiff')
        self.dlg.lineEdit_2.setText(filename)

    def select_output_file_3(self):
        """Connected to de third push button.
        """
        filename = QFileDialog.getSaveFileName(self.dlg,
        "Select output file ", "", '*.tiff')
        self.dlg.lineEdit_3.setText(filename)


    def run(self):
        """Run method that performs all the real work"""
        #Prepare dialog, cleaning
        self.clear_dialog()
        #Fill the combos
        layer_list = self.get_layers()
        #Plugin needs at least a raster layer
        init_time = 0
        if layer_list == []:
            error_dialog("Please load rasters layers")
        else:
            self.fill_combo(self.dlg.comboBox, layer_list, False)
            self.fill_combo(self.dlg.comboBox_2, layer_list, True)
            # show the dialog
            self.dlg.show()
            # Run the dialog event loop
            result = self.dlg.exec_()
            # See if OK was pressed
            if result == 1:
                init_time = time()

                #Flags
                qua_used = False
```

```
save_mask = False
save_resume = False

# Get input raster stack
selected_layer_index = self.dlg.comboBox.currentIndex()
time_serie_layer = self.dlg.layers[selected_layer_index]

# Get iterations number
iter_num = self.dlg.spinBox.value()

# Get output path
output_filename = self.dlg.lineEdit.text()

# Get quality stack if selected
selected_layer_index = self.dlg.comboBox_2.currentIndex()
if selected_layer_index != 0:
    quality_layer = self.dlg.layers[selected_layer_index - 1]
    qua_used = True

# Get path to save the quality mask
mask_filename = self.dlg.lineEdit_2.text()
if mask_filename != "":
    save_mask = True

# Get  path to save resume mask
resume_filename = self.dlg.lineEdit_3.text()
if resume_filename != "":
    save_resume = True

# Only quality mask is used to detect outlier?
use_sg = self.dlg.checkBox_2.isChecked()

#MAIN ALGORITHM

my_filepath = time_serie_layer.dataProvider().dataSourceUri()
ds = gdal.Open(my_filepath)
# Input info
band_count = time_serie_layer.bandCount()
band = np.array(ds.GetRasterBand(1).ReadAsArray())
rows = band.shape[0]
cols = band.shape[1]

# Creates raster a ndarray where the result is save.
bands = []
for h in range(1, band_count+1):
    bands.append(ds.GetRasterBand(h).ReadAsArray())
ndvi_raster = np.dstack(bands)

# Creates raster a ndarray where the mask result is save.
if qua_used:
    qua_filepath = quality_layer.dataProvider().dataSourceUri()
    ds = gdal.Open(qua_filepath)

    bands = []
    for h in range(1, band_count+1):
        bands.append(ds.GetRasterBand(h).ReadAsArray())
    qua_raster = np.dstack(bands)

#If resume need to be save
if save_resume:
    res_mask = np.zeros(shape=(rows, cols))

progree_bar = ProgressBar(self.iface, rows * cols)

# This funtion process quality data
def is_useful(num):
    """Extract uselfulness from num"""
    bin_value = format(num, '016b')
```

```
        # Extract usefulness value
        usef = bin_value[10:14]
        if (usef not in ['1101', '1110', '111']):
            #Useful
            return 1
        else:
            #Not Useful
            return 0
    is_useful = np.vectorize(is_useful)

    # Process each time serie in the image
    for i in range(rows):
        for j in range(cols):
            # Get time serie
            time_serie = TimeSerie(ndvi_raster[i][j])

            #Method 1 selected:
            if qua_used:
                # Get quality serie
                # and convert to a usefulness vector
                useful_serie = is_useful(qua_raster[i][j])
                qua_raster[i][j] = useful_serie
                if save_resume:
                    # Save resume data
                    res_mask[i][j] = len(useful_serie) - useful_serie.sum()
                # interpolate not useful data
                for n in range(len(useful_serie)):
                    if useful_serie[n] == 0:
                        time_serie.interpolation(n)

            #Method 2 selected:
            if use_sg:
                for _ in range(iter_num):
                    time_serie.ajust_to_trend()

            #Save data
            ndvi_raster[i][j] = time_serie.array

            # Aument progress bar
            progree_bar.make_progress()
            QtGui.qApp.processEvents()

    self.iface.messageBar().clearWidgets()
    self.iface.messageBar().pushMessage("Saving results...",
                            level=QgsMessageBar.INFO)

    driver = gdal.GetDriverByName("GTiff")
    # Output Stack
    output = driver.Create(output_filename, cols, rows, band_count,
                           gdal.GDT_Int16)
    if output is None:
        error_dialog("Unable to create raster " + output_filename)
    output.SetGeoTransform(ds.GetGeoTransform())
    output.SetProjection(ds.GetProjection())
    for b in range(band_count):
        output.GetRasterBand(b+1).WriteArray(ndvi_raster.take(b, axis=2))
    output.FlushCache()
    del output
    string_to_raster(output_filename)

    # Mask Output
    if save_mask:
        mask_output = driver.Create(mask_filename, cols, rows,
                                    band_count, gdal.GDT_Int16)
        if mask_output is None:
            error_dialog("Unable to create " + mask_filename)
        mask_output.SetGeoTransform(ds.GetGeoTransform())
        mask_output.SetProjection(ds.GetProjection())
```

```
    for b in range(band_count):
        mask_output.GetRasterBand(b+1).WriteArray(qua_raster.take(b, axis=2))
    mask_output.FlushCache()
    del mask_output
    string_to_raster(mask_filename)

# Resume Mask
if save_resume:
    resume_output = driver.Create(resume_filename, cols, rows,
                                  1, gdal.GDT_Int16)
    if resume_output is None:
        error_dialog("Unable to create " + resume_filename)
    resume_output.SetGeoTransform(ds.GetGeoTransform())
    resume_output.SetProjection(ds.GetProjection())
    resume_output.GetRasterBand(1).WriteArray(res_mask)
    resume_output.FlushCache()
    del resume_output
    string_to_raster(resume_filename)

final_time = time()
self.iface.messageBar().clearWidgets()
self.iface.messageBar().pushMessage
('Processing time: {0:.2f} min.'.format((final_time - init_time)/60.0),
                        level=QgsMessageBar.INFO)
```

A.2. time_series.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
TimeSeries

Class representing a time serie
                             -------------------
        begin                : 2017-02-22
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
 ***************************************************************************/
"""
from scipy.signal import savgol_filter
import numpy as np

class TimeSerie:
    """Abstraction of a time serie"""
    def __init__(self, serie_array):
        self.array = serie_array
        self.size = len(self.array)

    def interpolation(self, k):
        """interpolate k-value
        """
        if k == 0:
            inter = self.array[k+1]
        elif k == self.size - 1:
            inter = self.array[k-1]
        else:
            inter = (self.array[k-1] + self.array[k+1])/2.0
        self.array[k] = inter

    def ajust_to_trend(self):
        """Calculates a trend curve using savi_gol
        and interpolates value below the curve
        """
        savgol_serie = savgol_filter(self.array, 9, 6)
        self.array = np.where(self.array > savgol_serie,
                              self.array, savgol_serie)
```

A.3. outlier_eliminator_dialog.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 OutliersEliminatorDialog
                                 A QGIS plugin
 Eliminates atypicals values of MODIS raster stacks
                             -------------------
        begin                : 2017-03-02
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or modify
 *
 *   it under the terms of the GNU General Public License as published by
 *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *
 ***************************************************************************/
"""

import os
import gdal
import numpy as np
from utils import error_dialog

from PyQt4 import QtGui, uic

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'outliers_eliminator_dialog_base.ui'))


class OutliersEliminatorDialog(QtGui.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(OutliersEliminatorDialog, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)
        self.layers = []

    def input_check(self):
        """Validate user inputs
        """
        # Input Info
        selected_layer_index = self.comboBox.currentIndex()
        time_serie_layer = self.layers[selected_layer_index]
        my_filepath = time_serie_layer.dataProvider().dataSourceUri()
        ds = gdal.Open(my_filepath)
        band_count = time_serie_layer.bandCount()
        band = np.array(ds.GetRasterBand(1).ReadAsArray())
        if band_count < 23:
            error_dialog("Band number must be greather than 23")
            return False
```

```
        rows = band.shape[0]
        cols = band.shape[1]
        transform = ds.GetGeoTransform()

        # Output info
        output_filename = self.lineEdit.text()
        if output_filename == '':
            error_dialog("Select output path")
            return False

        #Quality Info
        qua_used = False
        selected_layer_index = self.comboBox_2.currentIndex()
        if selected_layer_index != 0:
            qua_used = True
            qua_layer = self.layers[selected_layer_index - 1]
            my_filepath = qua_layer.dataProvider().dataSourceUri()
            ds = gdal.Open(my_filepath)
            #Informacion de la capa de calidad.
            qua_band_count = qua_layer.bandCount()
            band = np.array(ds.GetRasterBand(1).ReadAsArray())
            qua_rows = band.shape[0]
            qua_cols = band.shape[1]
            qua_transform = ds.GetGeoTransform()
            #Ambas capas deben ser compatibles
            if band_count != qua_band_count or rows != qua_rows\
            or cols != qua_cols or transform != qua_transform:
                error_dialog("Input and quality stacks are incompatible")
                return False

        #Masks Info
        # If only quality stack must be used for detection
        # must have a quality layer selected
        only_qua = self.checkBox.isChecked()
        if selected_layer_index == 0 and only_qua:
            error_dialog("Select a quality stack to detect outlier")
            return False

        # If want to save a mask must have a quality stack selected
        mask_filename = self.lineEdit_2.text()
        resume_filename = self.lineEdit_3.text()
        if not qua_used:
            if mask_filename != '' or resume_filename != '':
                error_dialog("Select a quality stack to detect outlier")
                return False

        if not self.checkBox.isChecked() and not self.checkBox_2.isChecked():
            error_dialog("Please, select a method.")
            return False
        return True

    def accept(self):
        """Validates the dialog
        """
        valid_input = self.input_check()
        if valid_input:
            self.done(1) # Only accept the dialog if all inputs are valid
```

A.4. utils.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
utils
Contains useful functions for the development of the plugin.

Functions:
            -- string_to_raster: Open a raster layer in Qgis.
            -- error_dialog : Show error message with the given string
                                  -------------------
from:
http://gis.stackexchange.com/questions/144058
/how-to-load-a-raster-layer-using-pyqgis/144066
 ***************************************************************************/
"""
from PyQt4.QtCore import (QFileInfo, Qt)
from PyQt4.QtGui import (QMessageBox, QProgressBar)
from qgis.core import (QgsMapLayerRegistry, QgsRasterLayer)

def string_to_raster(raster):
    """Open a raster layer in Qgis

        :param raster: Name of the file to open as a raster layer
        :type raster: String

        :returns: returns 1 if successful, -1 otherwise
        :rtype: int
    """
    # Check if string is provided
    file_info = QFileInfo(raster)
    path = file_info.filePath()
    base_name = file_info.baseName()

    layer = QgsRasterLayer(path, base_name)
    QgsMapLayerRegistry.instance().addMapLayer(layer)

    if layer.isValid() is True:
        #Layer was loaded successfully
        return 1
    else:
        #Unable to read basename and file path"
        #Your string is probably invalid"
        return -1

def error_dialog(string):
    """Show error message with the given string
    """
    QMessageBox.information(None, "Error", string)

class ProgressBar(object):
    """docstring for ProgressBar"""
    def __init__(self, iface, max_p):
        super(ProgressBar, self).__init__()
        self.c = 0
        progress_bar = iface.messageBar().createMessage("Processing...")
        self.progress = QProgressBar()
        self.progress.setMaximum(max_p)
        self.progress.setAlignment(Qt.AlignLeft|Qt.AlignVCenter)
        progress_bar.layout().addWidget(self.progress)
        iface.messageBar().pushWidget(progress_bar, iface.messageBar().INFO)

    def make_progress(self):
        """Auments progress in the bar"""
        self.progress.setValue(self.c)
        self.c += 1
```

Apéndice B

Código plugin de filtrado

Principales archivos del plugin:

B.1. time_series_filters.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TimeSeriesFilters
                                 A QGIS plugin
 Filters for time series in a raster stack.
                              -------------------
        begin                : 2017-02-28
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or modify
 *
 *   it under the terms of the GNU General Public License as published by
 *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *
 ***************************************************************************/
"""
from PyQt4 import QtCore
from PyQt4 import QtGui
from PyQt4.QtCore import (QSettings, QTranslator, qVersion, QCoreApplication,
                          QDate)
from PyQt4.QtGui import (QAction, QIcon, QFileDialog)
from qgis.gui import QgsMessageBar
import gdal
import numpy as np
from utils import (error_dialog, string_to_raster, ProgressBar)
from time_serie import TimeSerie
from time import time
```

```
# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
from time_series_filters_dialog import TimeSeriesFiltersDialog
import os.path


class TimeSeriesFilters:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'TimeSeriesFilters_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)

        # Create the dialog (after translation) and keep reference
        self.dlg = TimeSeriesFiltersDialog()

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Time_Series_Filters')
        # TODO: We are going to let the user set this up in a future iteration
        self.toolbar = self.iface.addToolBar(u'TimeSeriesFilters')
        self.toolbar.setObjectName(u'TimeSeriesFilters')

        #Save as, buttons funtionality
        self.dlg.lineEdit.clear()
        self.dlg.pushButton.clicked.connect(self.select_output_file)

        self.dlg.checkBox.stateChanged.connect(self.check_savgol)
        self.dlg.checkBox_3.stateChanged.connect(self.check_dl)

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('TimeSeriesFilters', message)
```

```
def add_action(
    self,
    icon_path,
    text,
    callback,
    enabled_flag=True,
    add_to_menu=True,
    add_to_toolbar=True,
    status_tip=None,
    whats_this=None,
    parent=None):
    """Add a toolbar icon to the toolbar.

    :param icon_path: Path to the icon for this action. Can be a resource
        path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
    :type icon_path: str

    :param text: Text that should be shown in menu items for this action.
    :type text: str

    :param callback: Function to be called when the action is triggered.
    :type callback: function

    :param enabled_flag: A flag indicating if the action should be enabled
        by default. Defaults to True.
    :type enabled_flag: bool

    :param add_to_menu: Flag indicating whether the action should also
        be added to the menu. Defaults to True.
    :type add_to_menu: bool

    :param add_to_toolbar: Flag indicating whether the action should also
        be added to the toolbar. Defaults to True.
    :type add_to_toolbar: bool

    :param status_tip: Optional text to show in a popup when mouse pointer
        hovers over the action.
    :type status_tip: str

    :param parent: Parent widget for the new action. Defaults None.
    :type parent: QWidget

    :param whats_this: Optional text to show in the status bar when the
        mouse pointer hovers over the action.

    :returns: The action that was created. Note that the action is also
        added to self.actions list.
    :rtype: QAction
    """
    icon = QIcon(icon_path)
    action = QAction(icon, text, parent)
    action.triggered.connect(callback)
    action.setEnabled(enabled_flag)

    if status_tip is not None:
        action.setStatusTip(status_tip)

    if whats_this is not None:
        action.setWhatsThis(whats_this)

    if add_to_toolbar:
        self.toolbar.addAction(action)

    if add_to_menu:
        self.iface.addPluginToRasterMenu(
            self.menu,
```

```
            action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/TimeSeriesFilters/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'Filters for time series'),
            callback=self.run,
            parent=self.iface.mainWindow())


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginRasterMenu(
                self.tr(u'&Time Series Filters'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    def select_output_file(self):
        """Connected to de first push button.
        """
        filename = QFileDialog.getSaveFileName(self.dlg,
        "Select output file ", "", '*.tiff')
        self.dlg.lineEdit.setText(filename)

    def check_savgol(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.spinBox.setEnabled(True)
            self.dlg.spinBox_2.setEnabled(True)
        else:
            self.dlg.spinBox.setEnabled(False)
            self.dlg.spinBox_2.setEnabled(False)

    def check_dl(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.dateEdit.setEnabled(True)
            self.dlg.dateEdit_2.setEnabled(True)
            self.dlg.checkBox_2.setEnabled(True)
        else:
            self.dlg.dateEdit.setEnabled(False)
            self.dlg.dateEdit_2.setEnabled(False)
            self.dlg.checkBox_2.setEnabled(False)

    def clear_dialog(self):
        """
        Clean all el element on the main dialog
        """
        self.dlg.comboBox.clear()
        self.dlg.checkBox.setChecked(True)
        self.dlg.spinBox.setValue(3)
        self.dlg.spinBox_2.setValue(1)
        self.dlg.checkBox_3.setChecked(False)
        self.dlg.lineEdit.clear()
        self.dlg.dateEdit.setDate(QDate.currentDate())
        self.dlg.dateEdit_2.setDate(QDate.currentDate())
        self.dlg.dateEdit.setEnabled(False)
        self.dlg.dateEdit_2.setEnabled(False)
        self.dlg.checkBox_2.setEnabled(False)
        self.dlg.checkBox_2.setChecked(False)
```

```
def get_layers(self):
    """
    Reads the layer to be load in the combos boxs.
    """
    layers = self.iface.legendInterface().layers()
    layer_list = []
    for layer in layers:
        if layer.type() == 1: #Layer is raster
            layer_list.append(layer.name())
    self.dlg.layers = layers
    return layer_list


def run(self):
    """Run method that performs all the real work"""
    self.clear_dialog()
    layer_list = self.get_layers()
    if layer_list == []:
        error_dialog("Please load rasters layers")
    else:
        #fill Combo box
        self.dlg.comboBox.addItems(layer_list)
        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result == 1:
            init_time = time()
            # Get input raster stack
            selected_layer_index = self.dlg.comboBox.currentIndex()
            time_serie_layer = self.dlg.layers[selected_layer_index]
            # Get filters
            use_sav_gol = self.dlg.checkBox.isChecked()
            use_doble_logis = self.dlg.checkBox_3.isChecked()
            # Get Savitzky-Golay parameters if used
            if use_sav_gol:
                # Get windows legth
                win_length = self.dlg.spinBox.value()
                # Get polyorder value
                polyorder = self.dlg.spinBox_2.value()
            # Get output path
            output_filename = self.dlg.lineEdit.text()

            star_date = self.dlg.dateEdit.date()
            end_date = self.dlg.dateEdit_2.date()

            mask_checked = self.dlg.checkBox_2.isChecked()

            years = end_date.year() - star_date.year()

            # MAIN ALGORITHM
            my_filepath = time_serie_layer.dataProvider().dataSourceUri()
            ds = gdal.Open(my_filepath)
            # Input info
            band_count = time_serie_layer.bandCount()
            driver = gdal.GetDriverByName("GTiff")
            band = np.array(ds.GetRasterBand(1).ReadAsArray())
            rows = band.shape[0]
            cols = band.shape[1]
            # Creates raster a ndarray where the result is save.
            bands = []
            for h in range(1, band_count+1):
                bands.append(ds.GetRasterBand(h).ReadAsArray())
            raster = np.dstack(bands)
            # Create mask stack if needed
            if mask_checked:
```

```
    mask_stack = [np.full((rows, cols), None,
                          dtype=np.float32)] * (years-1)
    mask_stack = np.dstack(mask_stack)
#Progress bar
progree_bar = ProgressBar(self.iface, rows * cols)
# Process each time serie in the image
for i in range(rows):
    for j in range(cols):
        time_serie = TimeSerie(raster[i][j], mask_checked)
        if use_sav_gol:
            # Use Savitzky-Golay filter
            new_time_serie = time_serie.savgol(win_length,
                                               polyorder)
        if use_doble_logis:
            # Use Doble Logistic filter
            new_time_serie = time_serie.doble_logistic_filter(years)
            if mask_checked:
                mask_stack[i][j] = time_serie.mask

        raster[i][j] = new_time_serie
        # Aument progress bar
        progree_bar.make_progress()
        QtGui.qApp.processEvents()

self.iface.messageBar().clearWidgets()
self.iface.messageBar().pushMessage("Saving results...",
                        level=QgsMessageBar.INFO)
# Output Stack
output = driver.Create(output_filename, cols, rows, band_count,
                       gdal.GDT_Int32)
if output is None:
    error_dialog("Unable to create raster " + output_filename)
output.SetGeoTransform(ds.GetGeoTransform())
output.SetProjection(ds.GetProjection())
for b in range(band_count):
    output.GetRasterBand(b+1).WriteArray(raster.take(b, axis=2))
output.FlushCache()
del output
string_to_raster(output_filename)

if mask_checked:
    # Mask Output
    if output_filename.endswith('.tiff'):
        output_filename2 = output_filename[:-5]
    output_filename2 += "_mask.tiff"
    output2 = driver.Create(output_filename2, cols, rows,
                            years-1, gdal.GDT_Int16)
    if output2 is None:
        error_dialog("Unable to create mask raster")
    output2.SetGeoTransform(ds.GetGeoTransform())
    output2.SetProjection(ds.GetProjection())
    for b in range(years-1):
        output2.GetRasterBand(b+1).WriteArray(mask_stack.take(b, axis=2))
    output2.FlushCache()
    del output2
    string_to_raster(output_filename2)
self.iface.messageBar().clearWidgets()
```

B.2. time_series.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
TimeSeries

Class representing a time serie and its filters.
                             -------------------
        begin                : 2017-02-22
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
 ***************************************************************************/
"""
from scipy.signal import savgol_filter
import numpy as np
from scipy.optimize import curve_fit


def smooth_step(size):
    """Return a vector which start with zeros,
    between edge1 and edge2 goes from 0 to 1 progressively,
    and end with ones.
    """
    edge1 = size/2 - 4
    edge2 = size/2 + 4
    x = np.arange(size)
    x = np.clip((x-edge1) / float(edge2-edge1), 0.0, 1.0)
    return  1 - x*x*(3 - 2*x)

def get_ip(data ,curv_type):
    """Find left and right inflection points of data.
    """
    diff = np.diff(data)
    diff_sign = np.sign(diff)

    #Alrededor de un min:
    if curv_type == 0:
        argmin = data.argmin()
        for i in xrange(argmin, 0, -1):
            if abs(diff[i-1]) < abs(diff[i]) and diff_sign[i-1] == diff_sign[i]:
                i -= 1
                break
        for j in xrange(argmin, data.size):
            if abs(diff[j]) > abs(diff[j+1]) and diff_sign[j] == diff_sign[j+1]:
                j += 1
                break
    #Alrededor de un max
    else:
        argmax = data.argmax()
        for i in xrange(argmax):
```

```
            if abs(diff[i]) > abs(diff[i+1]) and diff_sign[i] == diff_sign[i+1]:
                i += 1
                break
        for j in xrange(data.size-2, argmax, -1):
            if abs(diff[j]) > abs(diff[j-1]) and diff_sign[j] == diff_sign[j+1]:
                j -= 1
                break
    return i, j

def fun_f(t, c1, c2, x1, x2, x3, x4):
    """Model funtion.
    """
    g = 1/(1 + np.exp((x1-t) / x2)) - 1/(1 + np.exp((x3-t) / x4))
    return c1 + c2*g

def fit_data(data, data_type):
    """Fit data to model funtion "fun_f"
    """
    last = data.size-1

    c1 = data.min() #Basal level
    c2 = data.max()-c1 #Amplitude
    #Alrededor de un min:
    if data_type == 0:
        x1, x3 = get_ip(data, 0) # Inflection points
        x2 = (data[0]-data[x1]) / float(x1) / 100.0 # Left change rate
        x4 = (data[last]-data[x3]) / float(last-x3) / 100.0 # Right change rate
    #Alrededor de un max:
    else:
        x1, x3 = get_ip(data, 1) # Inflection points
        x2 = ((data[x1]-data[0]) / float(x1)) / 100.0 # Left change rate
        x4 = ((data[x3]-data[last]) / float(last-x3)) / 100.0 # Right change rate
    data_t = np.arange(len(data))

    popt, pcov, infodict, errmsg, ier = curve_fit(fun_f, data_t, data,
                                                  p0=[c1, c2, x1, x2, x3, x4],
                                                  full_output=True,
                                                  maxfev=500,
                                                  xtol=0.005)
    return fun_f(data_t, *popt)

def fit_marge(l_curv_fit, c_curv_fit, r_curv_fit, min1, min2, max1, max2):
    """Marge local funtions in a global funtion.
    """
    l_r_curv_fit = l_curv_fit[min1-max1:]
    c_l_curv_fit = c_curv_fit[:max2-min1]

    smooth_vec = smooth_step(l_r_curv_fit.size)

    res_l = smooth_vec*l_r_curv_fit + (1-smooth_vec)*c_l_curv_fit

    c_r_curv_fit = c_curv_fit[max2-min1:]
    r_l_curv_fit = r_curv_fit[:min2-max2]

    smooth_vec = smooth_step(c_r_curv_fit.size)

    res_r = smooth_vec*c_r_curv_fit + (1-smooth_vec)*r_l_curv_fit

    return np.concatenate((res_l, res_r))

class TimeSerie:
    """Abstraction of a time serie"""
    def __init__(self, serie_array, get_mask):
        self.array = serie_array
        self.size = len(self.array)
        self.get_mask = get_mask
        self.mask = None
```

```
def savgol(self, win, poly):
    """Calculates Savitzky-Golay filter
    :param win: The length of the filter window.
    window_length must be a positive odd integer.
    :type win: int
    :param poly: The order of the polynomial used to fit the samples.
    polyorder must be less than window_length.
    """
    self.array = savgol_filter(self.array, win, poly)
    return self.array

def doble_logistic_filter(self, years):
    ndvi_data = self.array
    res = np.zeros(ndvi_data.size)

    try:
        minima = np.full(years, -1)
        for i in range(years):
            minima[i] = ndvi_data[i*23:(i+1)*23].argmin() + i*23
        maxima = np.full(years+1, -1)
        maxima[0] = ndvi_data[0:minima[0]].argmax()
        for i in range(years-1):
            maxima[i+1] = ndvi_data[minima[i]:minima[i+1]].argmax()+minima[i]
        maxima[years] = ndvi_data[minima[years-1]:].argmax() + minima[years-1]
    except:
        res = ndvi_data
    if self.get_mask:
        min_error = np.zeros(years)
        max_error = np.zeros(years-1)

    #Calculo los ajustes para las curvas locales con centro en el minimo
    min_fits = []
    for i in range(maxima.size-1):
        try:
            min_fits.append(fit_data(ndvi_data[maxima[i]:maxima[i+1]], 0))
        except:
            min_fits.append(ndvi_data[maxima[i]:maxima[i+1]])
            if self.get_mask:
                min_error[i] = 1

    #Calculo los ajustes para las curvas locales con centro en el maximo
    max_fits = []
    for i in range(minima.size-1):
        try:
            max_fits.append(fit_data(ndvi_data[minima[i]:minima[i+1]], 1))
        except:
            max_fits.append(ndvi_data[minima[i]:minima[i+1]])
            if self.get_mask:
                max_error[i] = 2

    if self.get_mask:
        self.mask = np.zeros(years)
        self.mask = (min_error[:-1] + min_error[1:]) + max_error

    #Calculo los ajustes globales
    try:
        res[:minima[0]] = ndvi_data[:minima[0]]
        for i in range(maxima.size-2):
            min1 = minima[i]
            min2 = minima[i+1]
            max1 = maxima[i]
            max2 = maxima[i+1]
            res[min1:min2] = fit_marge(min_fits[i], max_fits[i], min_fits[i+1],
                                                                 min1, min2, max1, max2)
        res[min2:] = ndvi_data[min2:]
    except:
        res = ndvi_data
    return res
```

B.3. time_series_filter_dialog.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TimeSeriesFiltersDialog
                                 A QGIS plugin
 Filters for time series in a raster stack.
                             -------------------
        begin                : 2017-02-28
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
 ***************************************************************************/
"""

import os
from utils import error_dialog

from PyQt4 import QtGui, uic

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'time_series_filters_dialog_base.ui'))


class TimeSeriesFiltersDialog(QtGui.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(TimeSeriesFiltersDialog, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.layers = []
        self.setupUi(self)

    def input_check(self):
        """
        Check possible input errors.
        """
        # Input Info
        selected_layer_index = self.comboBox.currentIndex()
        time_serie_layer = self.layers[selected_layer_index]
        band_count = time_serie_layer.bandCount()

        if band_count < 50:
            error_dialog("Band number must be greather than 23")
            return False
        #Output Info
        output_filename = self.lineEdit.text()
        if output_filename == '':
```

```
            error_dialog("Select output path")
            return False

        if self.checkBox.isChecked():
            win_length = self.spinBox.value()
            polyorder = self.spinBox_2.value()
            if win_length%2 == 0:
                error_dialog("The length of the filter window must be" +
                "a positive odd integer.")
                return False
            if polyorder >= win_length:
                error_dialog("Polyorder must be less than window length.")
                return False

        if self.checkBox_3.isChecked():
            if not self.dateEdit.date().isValid():
                error_dialog("Please, select a valid star date.")
                return False
            if not self.dateEdit_2.date().isValid():
                error_dialog("Please, select a valid end date.")
                return False
            if self.dateEdit_2.date() <= self.dateEdit.date():
                error_dialog("End date must be greater than star date.")
                return False
            years = self.dateEdit_2.date().year() - self.dateEdit.date().year()
            if band_count < years * 23:
                error_dialog("Must have at least" +
                " {0} band for {1} years.".format(years * 23, years))
                return False


        if not self.checkBox.isChecked() and not self.checkBox_3.isChecked():
            error_dialog("Please, choose a filter.")
            return False

        return True

    def accept(self):
        """Validates the dialog
        """
        valid_input = self.input_check()
        if valid_input:
            self.done(1) # Only accept the dialog if all inputs are valid
```

B.4. utils.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
utils
Contains useful functions for the development of the plugin.

Functions:
            -- string_to_raster: Open a raster layer in Qgis.
            -- error_dialog : Show error message with the given string
            -- pixel_to_coordenates: Take a pixel point (x, y)
                 and gives its coordenates

                                 -------------------
string:to_raster from:
http://gis.stackexchange.com/questions/144058
/how-to-load-a-raster-layer-using-pyqgis/144066
 ***************************************************************************/
"""
from PyQt4.QtCore import (QFileInfo, Qt)
from PyQt4.QtGui import (QMessageBox, QProgressBar)
from qgis.core import (QgsMapLayerRegistry, QgsRasterLayer)

def string_to_raster(raster):
    """Open a raster layer in Qgis

        :param raster: Name of the file to open as a raster layer
        :type raster: String

        :returns: returns 1 if successful, -1 otherwise
        :rtype: int
    """
    # Check if string is provided
    file_info = QFileInfo(raster)
    path = file_info.filePath()
    base_name = file_info.baseName()

    layer = QgsRasterLayer(path, base_name)
    QgsMapLayerRegistry.instance().addMapLayer(layer)

    if layer.isValid() is True:
        #Layer was loaded successfully
        return 1
    else:
        #Unable to read basename and file path"
        #Your string is probably invalid"
        return -1

def error_dialog(string):
    """Show error message with the given string
    """
    QMessageBox.information(None, "Error", string)

def pixel_to_coordenates(x, y, transform):
    """Take a pixel point (x, y) and gives its coordenates
    """
    longitude = transform[1] * y + transform[2] * x + \
    transform[1] * 0.5 + transform[2] * 0.5 + transform[0]
    latitude = transform[4] * y + transform[5] * x + \
    transform[4] * 0.5 + transform[5] * 0.5 + transform[3]
    return longitude, latitude


class ProgressBar(object):
    """docstring for ProgressBar"""
    def __init__(self, iface, max_p):
```

```
        super(ProgressBar, self).__init__()
        self.c = 0
        progress_bar = iface.messageBar().createMessage("Processing...")
        self.progress = QProgressBar()
        self.progress.setMaximum(max_p)
        self.progress.setAlignment(Qt.AlignLeft|Qt.AlignVCenter)
        progress_bar.layout().addWidget(self.progress)
        iface.messageBar().pushWidget(progress_bar, iface.messageBar().INFO)

    def make_progress(self):
        """Moves the progress bar
        """
        self.progress.setValue(self.c)
        self.c += 1
```

Apéndice C

Código plugin de extracción de fenologías

Principales archivos del plugin:

C.1. time_series_phen.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TimeSeriesPhen
                                 A QGIS plugin
 Extract phenometrics from  MODIS time series
                              -------------------
        begin                : 2017-10-31
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Marcos Landi
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or modify
 *
 *   it under the terms of the GNU General Public License as published by
 *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *
 ***************************************************************************/
"""
from PyQt4 import QtGui
from PyQt4 import QtCore
from PyQt4.QtCore import (QSettings, QTranslator, qVersion, QCoreApplication,
                          QDate)
from PyQt4.QtGui import (QAction, QIcon, QFileDialog)
from qgis.gui import QgsMessageBar
```

```
from utils import error_dialog, ProgressBar, string_to_raster
from time import time
import gdal
import numpy as np
from time_serie import TimeSerie


# Initialize Qt resources from file resources.py
import resources
# Import the code for the dialog
from time_series_phen_dialog import TimeSeriesPhenDialog
import os.path



class TimeSeriesPhen:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'TimeSeriesPhen_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)

            if qVersion() > '4.3.3':
                QCoreApplication.installTranslator(self.translator)


        # Create the dialog (after translation) and keep reference
        self.dlg = TimeSeriesPhenDialog()

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Time_Series_Phenometrics')
        # TODO: We are going to let the user set this up in a future iteration
        self.toolbar = self.iface.addToolBar(u'TimeSeriesPhen')
        self.toolbar.setObjectName(u'TimeSeriesPhen')

        self.ds = None
        self.driver = None
        self.band_count = None
        self.rows = None
        self.cols = None

        #Save as, buttons funtionality
        self.dlg.lineEdit.clear()
        self.dlg.pushButton.clicked.connect(self.select_output_location)

        self.dlg.checkBox_7.stateChanged.connect(self.click_start_season)
        self.dlg.checkBox_8.stateChanged.connect(self.click_end_season)
```

```
        self.dlg.checkBox_12.stateChanged.connect(self.click_season_integ)
        self.dlg.checkBox_13.stateChanged.connect(self.click_all)
        self.dlg.checkBox_4.stateChanged.connect(self.click_inc)
        self.dlg.checkBox_5.stateChanged.connect(self.click_dec)

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.

        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('TimeSeriesPhen', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
```

```
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            self.toolbar.addAction(action)

        if add_to_menu:
            self.iface.addPluginToRasterMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

        icon_path = ':/plugins/TimeSeriesPhen/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'MODIS Time Series phenometrics extraction'),
            callback=self.run,
            parent=self.iface.mainWindow())


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginRasterMenu(
                self.tr(u'&Time Series Phenometrics'),
                action)
            self.iface.removeToolBarIcon(action)
        # remove the toolbar
        del self.toolbar

    def select_output_location(self):
        """Action of th epush button"""
        location = QFileDialog.getExistingDirectory(self.dlg,
        "Select outputs location ", os.getcwd(), QtGui.QFileDialog.ShowDirsOnly)
        self.dlg.lineEdit.setText(location)


    def clear_dialog(self):
        """
        Clear dialog before star.
        """
        self.dlg.comboBox.clear()
        self.dlg.lineEdit.clear()
        self.dlg.dateEdit.setDate(QDate.currentDate())
        self.dlg.dateEdit_2.setDate(QDate.currentDate())
        self.dlg.comboBox_2.setCurrentIndex(0)
        self.dlg.checkBox.setChecked(False)
        self.dlg.checkBox_2.setChecked(False)
        self.dlg.checkBox_3.setChecked(False)
        self.dlg.checkBox_4.setChecked(False)
        self.dlg.checkBox_5.setChecked(False)
```

```
        self.dlg.checkBox_6.setChecked(False)
        self.dlg.checkBox_7.setChecked(False)
        self.dlg.checkBox_8.setChecked(False)
        self.dlg.checkBox_9.setChecked(False)
        self.dlg.checkBox_10.setChecked(False)
        self.dlg.checkBox_11.setChecked(False)
        self.dlg.doubleSpinBox.setValue(10.00)
        self.dlg.doubleSpinBox_2.setValue(10.00)
        self.dlg.checkBox_12.setChecked(False)
        self.dlg.checkBox_13.setChecked(False)
        self.dlg.doubleSpinBox_3.setValue(5.00)
        self.dlg.doubleSpinBox_4.setValue(5.00)

    def click_start_season(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.doubleSpinBox.setEnabled(True)
        else:
            self.dlg.doubleSpinBox.setEnabled(False)

    def click_end_season(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.doubleSpinBox_2.setEnabled(True)
        else:
            self.dlg.doubleSpinBox_2.setEnabled(False)

    def click_inc(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.doubleSpinBox_3.setEnabled(True)
        else:
            self.dlg.doubleSpinBox_3.setEnabled(False)

    def click_dec(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.doubleSpinBox_4.setEnabled(True)
        else:
            self.dlg.doubleSpinBox_4.setEnabled(False)

    def click_season_integ(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.comboBox_2.setEnabled(True)
        else:
            self.dlg.comboBox_2.setEnabled(False)

    def click_all(self, state):
        if state == QtCore.Qt.Checked:
            self.dlg.checkBox.setChecked(True)
            self.dlg.checkBox_2.setChecked(True)
            self.dlg.checkBox_3.setChecked(True)
            self.dlg.checkBox_4.setChecked(True)
            self.dlg.checkBox_5.setChecked(True)
            self.dlg.checkBox_6.setChecked(True)
            self.dlg.checkBox_7.setChecked(True)
            self.dlg.checkBox_8.setChecked(True)
            self.dlg.checkBox_9.setChecked(True)
            self.dlg.checkBox_10.setChecked(True)
            self.dlg.checkBox_11.setChecked(True)
            self.dlg.checkBox_12.setChecked(True)
            self.dlg.checkBox_13.setChecked(True)
        else:
            self.dlg.checkBox.setChecked(False)
            self.dlg.checkBox_2.setChecked(False)
            self.dlg.checkBox_3.setChecked(False)
            self.dlg.checkBox_4.setChecked(False)
            self.dlg.checkBox_5.setChecked(False)
            self.dlg.checkBox_6.setChecked(False)
            self.dlg.checkBox_7.setChecked(False)
            self.dlg.checkBox_8.setChecked(False)
            self.dlg.checkBox_9.setChecked(False)
```

```
        self.dlg.checkBox_10.setChecked(False)
        self.dlg.checkBox_11.setChecked(False)
        self.dlg.checkBox_12.setChecked(False)
        self.dlg.checkBox_13.setChecked(False)


def get_layers(self):
    """
    Reads the layer to be load in the combos box.
    """
    layers = self.iface.legendInterface().layers()
    layer_list = []
    for layer in layers:
        if layer.type() == 1: #Layer is raster
            layer_list.append(layer.name())
    self.dlg.layers = layers
    return layer_list

def save_pheno(self, output_filename, raster, save_type, band_count):
    output = self.driver.Create(output_filename, self.cols, self.rows,
                                band_count, save_type)
    if output is None:
        error_dialog("Unable to create raster " + output_filename)
    output.SetGeoTransform(self.ds.GetGeoTransform())
    output.SetProjection(self.ds.GetProjection())
    for b in xrange(band_count):
        output.GetRasterBand(b+1).WriteArray(raster.take(b, axis=2))
    output.FlushCache()
    del output
    string_to_raster(output_filename)


def run(self):
    self.clear_dialog()
    layer_list = self.get_layers()
    if layer_list == []:
        error_dialog("Please load rasters layers")
    else:
        #fill Combo box
        self.dlg.comboBox.addItems(layer_list)
        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result == 1:
            init_time = time()
            # Get input raster stack
            time_serie_layer = self.dlg.layers[self.dlg.comboBox.currentIndex()]
            #Get location to save inputs
            output_location = self.dlg.lineEdit.text()
            # Get years to calculate
            start_date = self.dlg.dateEdit.date()
            end_date = self.dlg.dateEdit_2.date()
            years = end_date.year() - start_date.year() - 1

            # MAIN ALGORITHM
            my_filepath = time_serie_layer.dataProvider().dataSourceUri()
            self.ds = gdal.Open(my_filepath)

            # Input info
            band_count = time_serie_layer.bandCount()
            self.driver = gdal.GetDriverByName("GTiff")
            band = np.array(self.ds.GetRasterBand(1).ReadAsArray())
            self.rows = band.shape[0]
            self.cols = band.shape[1]

            # Creates raster ndarray of the input
```

```
bands = []
for h in range(1, band_count+1):
    bands.append(self.ds.GetRasterBand(h).ReadAsArray())
raster = np.dstack(bands)

# Get season base level
bas_checked = self.dlg.checkBox.isChecked()
if bas_checked:
    bas_stack = [np.full((self.rows, self.cols),
                         None, dtype=np.float32)] * years
    bas_stack = np.dstack(bas_stack)

# Get season maximum position
max_pos_checked = self.dlg.checkBox_3.isChecked()
if max_pos_checked:
    max_pos_stack = [np.full((self.rows, self.cols), 0,
                             dtype=np.int32)] * years
    max_pos_stack = np.dstack(max_pos_stack)

# Get season maximun value
max_value_checked = self.dlg.checkBox_2.isChecked()
if max_value_checked:
    max_value_stack = [np.full((self.rows, self.cols), None,
                               dtype=np.float32)] * years
    max_value_stack = np.dstack(max_value_stack)

# Get increase rate
inc_perc = self.dlg.doubleSpinBox_3.value() / 100.0
inc_checked = self.dlg.checkBox_4.isChecked()
if inc_checked:
    inc_stack = [np.full((self.rows, self.cols), None,
                         dtype=np.float32)] * years
    inc_stack = np.dstack(inc_stack)

# Get decrese rate
dec_perc = self.dlg.doubleSpinBox_4.value() / 100.0
dec_checked = self.dlg.checkBox_5.isChecked()
if dec_checked:
    dec_stack = [np.full((self.rows, self.cols), None,
                         dtype=np.float32)] * years
    dec_stack = np.dstack(dec_stack)

# Get ampitude
amp_checked = self.dlg.checkBox_6.isChecked()
if amp_checked:
    amp_stack = [np.full((self.rows, self.cols), None,
                         dtype=np.float32)] * years
    amp_stack = np.dstack(amp_stack)

# Get start of season
sos_perc = self.dlg.doubleSpinBox.value() / 100.0
sos_checked = self.dlg.checkBox_7.isChecked()
if sos_checked:
    sos_stack = [np.full((self.rows, self.cols), None,
                         dtype=np.float32)] * years
    sos_stack = np.dstack(sos_stack)

# Get end of season
eos_perc = self.dlg.doubleSpinBox_2.value() / 100.0
eos_checked = self.dlg.checkBox_8.isChecked()
if eos_checked:
    eos_stack = [np.full((self.rows, self.cols), None,
                         dtype=np.float32)] * years
    eos_stack = np.dstack(eos_stack)

# Get integral
integ_checked = self.dlg.checkBox_9.isChecked()
if integ_checked:
```

```
        integ_stack = [np.full((self.rows, self.cols), None,
                               dtype=np.float32)] * years
        integ_stack = np.dstack(integ_stack)

    # Get lenght
    length_checked = self.dlg.checkBox_10.isChecked()
    if length_checked:
        length_stack = [np.full((self.rows, self.cols), None,
                                dtype=np.float32)] * years
        length_stack = np.dstack(length_stack)
    # Get relative range
    range_checked = self.dlg.checkBox_11.isChecked()
    if range_checked:
        range_stack = [np.full((self.rows, self.cols), None,
                               dtype=np.float32)] * years
        range_stack = np.dstack(range_stack)

    # Get seasonal integral
    s_integ_checked = self.dlg.checkBox_12.isChecked()
    if s_integ_checked:
        s1_stack = [np.full((self.rows, self.cols), None,
                            dtype=np.float32)] * (years-1)
        s1_stack = np.dstack(s1_stack)

        s2_stack = [np.full((self.rows, self.cols), None,
                            dtype=np.float32)] * (years-1)
        s2_stack = np.dstack(s2_stack)

        s3_stack = [np.full((self.rows, self.cols), None,
                            dtype=np.float32)] * (years-1)
        s3_stack = np.dstack(s3_stack)

        s4_stack = [np.full((self.rows, self.cols), None,
                            dtype=np.float32)] * (years-1)
        s4_stack = np.dstack(s4_stack)


    pro_bar = ProgressBar(self.iface, self.rows * self.cols)

    # Calculate selected phenology
    for i in xrange(self.rows):
        for j in xrange(self.cols):
            s_start_date = '{0}{1}'.format(format(start_date.day(), '02'),
                           format(start_date.month(), '02'))
            time_serie = TimeSerie(raster[i][j], years, s_start_date)
            if bas_checked:
                bas_stack[i][j] = time_serie.get_base()

            if max_pos_checked:
                max_pos_stack[i][j] = time_serie.get_max_pos()

            if max_value_checked:
                max_value_stack[i][j] = time_serie.get_max_value()

            if inc_checked:
                inc_stack[i][j] = time_serie.get_inc(inc_perc)

            if dec_checked:
                dec_stack[i][j] = time_serie.get_dec(dec_perc)

            if amp_checked:
                amp_stack[i][j] = time_serie.get_amp()

            if sos_checked:
                sos_stack[i][j] = time_serie.get_sos(sos_perc)

            if eos_checked:
                eos_stack[i][j] = time_serie.get_eos(eos_perc)
```

```
                if integ_checked:
                    integ_stack[i][j] = time_serie.get_integ(sos_perc, eos_perc)

                if length_checked:
                    length_stack[i][j] = time_serie.get_length(sos_perc, eos_perc)

                if range_checked:
                    range_stack[i][j] = time_serie.get_range(sos_perc, eos_perc)

                if s_integ_checked:
                    s_integ_res = time_serie.get_s_integ()
                    s1_stack[i][j] = s_integ_res[0]
                    s2_stack[i][j] = s_integ_res[1]
                    s3_stack[i][j] = s_integ_res[2]
                    s4_stack[i][j] = s_integ_res[3]

                pro_bar.make_progress()
                QtGui.qApp.processEvents()

        self.iface.messageBar().clearWidgets()
        self.iface.messageBar().pushMessage("Saving results...",
                             level=QgsMessageBar.INFO)

        #Save selected phenology
        if bas_checked:
            self.save_pheno(output_location + '\Base_level.tiff',
                            bas_stack, gdal.GDT_Float32, years)
        if max_pos_checked:
            self.save_pheno(output_location + '\Maximun_positions.tiff',
                            max_pos_stack, gdal.GDT_Int32, years)

        if max_value_checked:
            self.save_pheno(output_location + '\Maximun_values.tiff',
                            max_value_stack, gdal.GDT_Float32, years)

        if inc_checked:
            self.save_pheno(output_location + '\Rate_of_increase.tiff',
                            inc_stack, gdal.GDT_Float32, years)

        if dec_checked:
            self.save_pheno(output_location + '\Rate_of_decrease.tiff',
                            dec_stack, gdal.GDT_Float32, years)

        if amp_checked:
            self.save_pheno(output_location + '\Amplitude.tiff',
                            amp_stack, gdal.GDT_Float32, years)

        if sos_checked:
            self.save_pheno(output_location + '\Start_of_season.tiff',
                            sos_stack, gdal.GDT_Float32, years)

        if eos_checked:
            self.save_pheno(output_location + '\End_of_season.tiff',
                            eos_stack, gdal.GDT_Float32, years)

        if integ_checked:
            self.save_pheno(output_location + '\Integral.tiff',
                            integ_stack, gdal.GDT_Float32, years)

        if length_checked:
            self.save_pheno(output_location + '\Length_of_season.tiff',
                            length_stack, gdal.GDT_Float32, years)

        if range_checked:
            self.save_pheno(output_location + '\Relative_range.tiff',
                            range_stack, gdal.GDT_Float32, years)
```

```
if s_integ_checked:
    if self.dlg.comboBox.currentIndex() == 0:
        self.save_pheno(output_location + '\Summer_integral.tiff',
                        s1_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Autumn_integral.tiff',
                        s2_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Winter_integral.tiff',
                        s3_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Spring_integral.tiff',
                        s4_stack, gdal.GDT_Float32, years-1)
    else:
        self.save_pheno(output_location + '\Winter_integral.tiff',
                        s1_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Spring_integral.tiff',
                        s2_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Summer_integral.tiff',
                        s3_stack, gdal.GDT_Float32, years-1)
        self.save_pheno(output_location + '\Autumn_integral.tiff',
                        s4_stack, gdal.GDT_Float32, years-1)
self.iface.messageBar().clearWidgets()
```

C.2. time_series.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
TimeSeries

Class representing a time serie and its filters.
                             -------------------
        begin                : 2017-02-22
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Landi Marcos
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
 ***************************************************************************/
"""

import numpy as np
import scipy.integrate as integrate


"""
Calendar date/Day of the year
"""
M_doy = {}
M_doy['0101'] = 1
M_doy['1701'] = 17
M_doy['0202'] = 33
M_doy['1802'] = 49
M_doy['0603'] = 65
M_doy['2203'] = 81
M_doy['0704'] = 97
M_doy['2304'] = 113
M_doy['0905'] = 129
M_doy['2505'] = 145
M_doy['1006'] = 161
M_doy['2606'] = 177
M_doy['1207'] = 193
M_doy['2807'] = 209
M_doy['1308'] = 225
M_doy['2908'] = 241
M_doy['1409'] = 257
M_doy['3009'] = 273
M_doy['1610'] = 289
M_doy['0111'] = 305
M_doy['1711'] = 321
M_doy['0312'] = 337
M_doy['1912'] = 353

class TimeSerie:
    def __init__(self, serie_array, years, start_date):
        self.array = serie_array
        self.size = len(self.array)
```

```
        self.years = years
        self.start_date = start_date
        self.usable = True

        self.min_t = None
        self.base = None
        self.max_pos = None
        self.max_value = None
        self.inc = None
        self.dec = None
        self.amp = None
        self.sos = None
        self.eos = None
        self.integ = None
        self.length = None
        self.range = None
        self.s1 = None
        self.s2 = None
        self.s3 = None
        self.s4 = None

    def get_min_t(self):
        """Time of minimum
        """
        if self.min_t == None:
            res_size = self.years + 1
            self.min_t = np.full(res_size, -1, dtype=np.int32)
            try:
                for i in xrange(res_size):
                    self.min_t[i] = self.array[i*23:(i+1)*23].argmin() + i*23
                if -1 in self.min_t:
                    self.usable = False
                    self.min_t = np.full(res_size, -1, dtype=np.int32)
            except:
                self.usable = False
                self.min_t = np.full(self.years, -1, dtype=np.int32)
        return self.min_t

    def get_base(self):
        """Base level
        """
        if self.min_t == None:
            self.get_min_t()
        if self.usable:
            minima = self.array[self.min_t]
            self.base = np.array([(minima[i] + minima[i+1])/2.0
                                  for i in xrange(minima.size-1)])
            self.base = self.base/10000.0
        else:
            self.base = np.full(self.years, -1, dtype=np.float32)
        return self.base

    def get_max_pos(self):
        """Maximun position
        """
        if self.min_t == None:
            self.get_min_t()
        if self.usable:
            self.max_pos = np.full(self.years, -1, dtype=np.int32)
            min_t = self.min_t
            for i in xrange(self.years):
                self.max_pos[i] = self.array[min_t[i]:min_t[i+1]].argmax() + min_t[i]
        else:
            self.max_pos = np.full(self.years, -1, dtype=np.int32)
        return self.max_pos

    def get_max_value(self):
        """Maximun value
```

```
        """
        if self.max_pos == None:
            self.get_max_pos()
        if self.usable:
            self.max_value = self.array[self.max_pos]/10000.0
        else:
            self.max_value = np.full(self.years, -1, dtype=np.float32)
        return self.max_value

    def get_inc(self, perc):
        """Increase rate
        """
        if self.min_t == None:
            self.get_min_t()
        if self.max_pos == None:
            self.get_max_pos()
        if self.usable:
            self.inc = np.full(self.years, 0, dtype=np.float32)
            for i in xrange(self.years):
                try:
                    data = self.array[self.min_t[i]:self.max_pos[i]]

                    p1_ndvi = data[0] + data[0] * perc
                    p1_pos = np.where(data > p1_ndvi)[0][0]

                    p2_ndvi = data[-1] - data[-1] * perc
                    p2_pos = np.where(data < p2_ndvi)[0][-1]

                    if p1_pos < p2_pos:
                        self.inc[i] = (data[p2_pos] - data[p1_pos])/10000.0/(p2_pos-p1_pos)
                    else:
                        self.inc[i] = None
                except:
                    self.inc = np.full(self.years, None, dtype=np.float32)

        else:
            self.inc = np.full(self.years, 0, dtype=np.float32)
        return self.inc

    def get_dec(self, perc):
        """Decrese rate
        """
        if self.min_t == None:
            self.get_min_t()
        if self.max_pos == None:
            self.get_max_pos()
        if self.usable:
            self.inc = np.full(self.years, 0, dtype=np.float32)
            for i in xrange(self.years):
                try:
                    data = self.array[self.max_pos[i]:self.min_t[i+1]]

                    p1_ndvi = data[0] - data[0] * perc
                    p1_pos = np.where(data < p1_ndvi)[0][0]

                    p2_ndvi = data[-1] + data[-1] * perc
                    p2_pos = np.where(data > p2_ndvi)[0][-1]

                    if p1_pos < p2_pos:
                        self.inc[i] = (data[p2_pos] - data[p1_pos])/10000.0/(p2_pos-p1_pos)
                    else:
                        self.inc[i] = None
                except:
                    self.inc = np.full(self.years, None, dtype=np.float32)

        else:
            self.inc = np.full(self.years, 0, dtype=np.float32)
        return self.inc
```

```
def get_amp(self):
    """Amplitude
    """
    if self.base == None:
        self.get_base()
    if self.max_value == None:
        self.get_max_value()
    if self.usable:
        self.amp = (self.max_value - self.base)
    else:
        self.amp = np.full(self.years, -1, dtype=np.float32)
    return self.amp

def get_sos(self, perc):
    """Start of season
    """
    if self.min_t == None:
        self.get_min_t()
    if self.usable:
        self.sos = np.full(self.years, -1, dtype=np.float32)
        for i in xrange(self.years):
            min1 = self.min_t[i]
            sos_ndvi = self.array[min1] + self.array[min1]*perc
            t = min1+1
            fail = False
            while self.array[t] < sos_ndvi:
                t += 1
                if t > self.min_t[i+1]:
                    fail = True
                    break
            if not fail:
                a = self.array[t] - self.array[t-1]
                b = self.array[t] - a*t
                self.sos[i] = (sos_ndvi-b)/a
            else:
                self.sos = np.full(self.years, -1, dtype=np.float32)
    else:
        self.sos = np.full(self.years, -1, dtype=np.float32)
    return self.sos

def get_eos(self, perc):
    """End of season
    """
    if self.min_t == None:
        self.get_min_t()
    if self.usable:
        self.eos = np.full(self.years, -1, dtype=np.float32)
        for i in xrange(self.years):
            min2 = self.min_t[i+1]
            eos_ndvi = self.array[min2] + self.array[min2]*perc
            t = min2 - 1
            fail = False
            while self.array[t] < eos_ndvi:
                t -= 1
                if t < self.min_t[i]:
                    fail = True
                    break
            if not fail:
                a = self.array[t+1] - self.array[t]
                b = self.array[t] - a*t
                self.eos[i] = (eos_ndvi-b)/a
            else:
                self.eos = np.full(self.years, -1, dtype=np.float32)
    else:
        self.eos = np.full(self.years, -1, dtype=np.float32)
    return self.eos
```

```
def get_length(self, sos_perc, eos_perc):
    """Lenght of season
    """
    self.length = np.full(self.years, -1, dtype=np.float32)
    if self.sos == None:
        self.get_sos(sos_perc)
    if self.eos == None:
        self.get_eos(eos_perc)
    if self.usable:
        self.length = self.eos - self.sos
    else:
        self.length = np.full(self.years, -1, dtype=np.float32)
    return self.length

def get_integ(self, sos_perc, eos_perc):
    """Integral of season
    """
    self.integ = np.full(self.years, -1, dtype=np.float32)
    if self.sos == None:
        self.get_sos(sos_perc)
    if self.eos == None:
        self.get_eos(sos_perc)
    if self.usable and self.array[0]:
        for i in range(self.years):
            try:
                if self.sos[i] != -1 and self.eos[i] != -1:
                    data = self.array[int(self.sos[i]):int(self.eos[i])]
                    self.integ[i] = np.trapz(data)/10000.0
                else:
                    self.integ[i] = -1
            except:
                self.integ[i] = -1
    else:
        self.integ = np.full(self.years, -1, dtype=np.float32)
    return self.integ

def get_range(self, sos_perc, eos_perc):
    """Relative range
    """
    self.range = np.full(self.years, -1, dtype=np.float32)
    if self.amp == None:
        self.get_amp()
    if self.integ == None:
        self.get_integ(sos_perc, eos_perc)
    if self.usable:
        #Ver que que pasa si la integral es negativa
        self.range = self.amp/self.integ
    else:
        self.range = np.full(self.years, -1, dtype=np.float32)
    return self.range

def get_s_integ(self):
    """Seasonal integral
    """
    self.s1 = np.full(self.years-1, -1, dtype=np.float32)
    self.s2 = np.full(self.years-1, -1, dtype=np.float32)
    self.s3 = np.full(self.years-1, -1, dtype=np.float32)
    self.s4 = np.full(self.years-1, -1, dtype=np.float32)

    try:
        last_doy = np.empty(self.years,dtype = 'int32')
        last_doy[0] = (353 - M_doy[self.start_date])/16.0
        for i in xrange(1, self.years):
            last_doy[i] = last_doy[i-1] + 23

        for i in xrange(self.years-1):
            data = self.array[last_doy[i]:last_doy[i+1]]
            self.s1[i] = np.trapz(data[0:6])/10000.0
```

```
            self.s2[i] = np.trapz(data[6:12])/10000.0
            self.s3[i] = np.trapz(data[12:18])/10000.0
            self.s4[i] = np.trapz(data[18:23])/10000.0
    except:
        self.s1 = np.full(self.years-1, -1, dtype=np.float32)
        self.s2 = np.full(self.years-1, -1, dtype=np.float32)
        self.s3 = np.full(self.years-1, -1, dtype=np.float32)
        self.s4 = np.full(self.years-1, -1, dtype=np.float32)

    return (self.s1, self.s2, self.s3, self.s4)
```

C.3. time_series_phen_dialog.py

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TimeSeriesPhenDialog
                                 A QGIS plugin
 Extract phenometrics from MODIS time series
                             -------------------
        begin                : 2017-10-31
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Bortagaray Natalia, Marcos Landi
        email                : natiborta@gmail.com
 ***************************************************************************/

/***************************************************************************
 *
 *
 *   This program is free software; you can redistribute it and/or modify
 *
 *   it under the terms of the GNU General Public License as published by
 *
 *   the Free Software Foundation; either version 2 of the License, or
 *
 *   (at your option) any later version.
 *
 *
 *
 ***************************************************************************/
"""

import os

from PyQt4 import QtGui, uic
from utils import error_dialog

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'time_series_phen_dialog_base.ui'))


class TimeSeriesPhenDialog(QtGui.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        """Constructor."""
        super(TimeSeriesPhenDialog, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.layers = []

        self.setupUi(self)

    def input_check(self):
        """
        Check possible input errors.
        """
        # Input Info
        selected_layer_index = self.comboBox.currentIndex()
        time_serie_layer = self.layers[selected_layer_index]
        band_count = time_serie_layer.bandCount()
        if band_count < 46:
            error_dialog("Band number must be greather than 46")
            return False
        #Output Info
        output_location = self.lineEdit.text()
        if output_location == '':
```

```
            error_dialog("Select output location")
            return False

        #Date info
        if not self.dateEdit.date().isValid():
            error_dialog("Please, select a valid star date.")
            return False

        if not self.dateEdit_2.date().isValid():
            error_dialog("Please, select a valid end date.")
            return False

        if self.dateEdit_2.date() <= self.dateEdit.date():
            error_dialog("End date must be greater than star date.")
            return False

        years = self.dateEdit_2.date().year() - self.dateEdit.date().year()
        if band_count < years * 23:
            error_dialog("Must have at least" +
            "{0} band for {1} years.".format(years * 23, years))
            return False

        return True

    def accept(self):
        """
        Validates the dialog
        """
        valid_input = self.input_check()
        if valid_input:
            self.done(1) # Only accept the dialog if all inputs are valid
```