



Universidad Nacional Experimental del Táchira

Vicerectorado Académico

Decanato de Docencia

Departamento de Ingeniería en Informática

Trabajo de aplicación profesional

Proyecto especial de grado

PAQUETE EN LENGUAJE R, PARA LA SIMULACIÓN DEL MODELO MOMOS

Autor: Ghabriel Villarreal

C.I.:22.676.954

ghabriel.villarreal@unet.edu.ve

Tutor(es): Rossana Timaure

rttg@unet.edu.ve

San Cristóbal, Octubre de 2022

Índice general

1. El Problema	6
1.1. Planteamiento y formulación del problema	6
1.2. Objetivos	8
1.2.1. Objetivo General	8
1.2.2. Objetivos Específicos	8
1.3. Justificación e Importancia	8
1.4. Alcance y Limitaciones	8
2. Fundamentos Teóricos	9
2.1. Antecedentes	9
2.2. Bases Teóricas	10
2.2.1. Modelos	10
2.2.2. Lenguaje R.	12
2.2.3. RStudio.	12
2.2.4. Algoritmos en R/RStudio.	13
2.2.5. Estructura de paquetes en R/RStudio.	14
2.2.6. Pruebas estándar y pruebas funcionales	16
2.2.7. Glosario	18
2.3. <i>GNU General Public License v2.0 (GPL-2.0)</i>	20
3. Fundamentos Metodológicos	21
3.1. Enfoque de la investigación	21
3.2. Tipo o nivel de investigación	21
3.3. Diseño de la investigación	22
3.4. Metodología	23
3.5. Aspectos administrativos	24

4. Desarrollo	25
4.1. Creación del esqueleto del paquete	25
4.2. Registrar el método para el envío y uso de funciones	29
4.3. Diseño y codificación de las funciones	32
4.4. Pruebas unitarias de las funciones	41
4.5. Chequear la carga del paquete	43
4.6. Construcción del método de distribución del paquete	43
5. Conclusiones y Recomendaciones	46
5.1. Conclusiones	46
5.2. Recomendaciones	46

Índice de figuras

3.1. Diagrama de Gantt con la planificación del proyecto especial de grado	24
4.1. Nuevo proyecto	27
4.2. Nuevo directorio	27
4.3. Tipo de proyecto	28
4.4. Nombre del paquete y directorio de trabajo	28
4.5. Estructura archivo DESCRIPTION	29
4.6. Esqueleto del paquete	29
4.7. Datos de salida sin calibrar	31
4.8. Datos de salida calibrados	32
4.9. Establecer parámetros y constantes	33
4.10. Establecer la función inicial y	33
4.11. Función derivación	34
4.12. Ecuación diferencial	35
4.13. Resultado de la ecuación diferencial	36
4.14. Extracción de datos experimentales	37
4.15. Creación del vector de residuos	37
4.16. Encontrar el valor ideal del parámetro Kresp	38
4.17. Valor ideal del parámetro Kresp	38
4.18. Dataframe con los datos calibrados de MOMOS	39
4.19. Método para graficar los resultados obtenidos	40
4.20. Gráfica de los resultados obtenidos	41
4.21. Resultado de las pruebas unitarias del paquete	42
4.22. Cobertura de las pruebas sobre el código creado	42
4.23. Resultado del chequeo del paquete	43
4.24. Resultado de la compilación del binario del paquete	44

4.25. Resultado de la compilación del código fuente del paquete	45
4.26. Código fuente y binario del paquete	45

Capítulo 1

El Problema

1.1. Planteamiento y formulación del problema

La materia orgánica estable del suelo (MOS) se caracteriza por ser el resultado del proceso de descomposición de los residuos de animales y vegetales, los cuales varían en proporción y estado. Estos residuos son sustancias que contribuyen a la fertilidad del suelo, por lo que, la materia orgánica del suelo MOS, viene a ser uno de los factores de importancia que determina localidad de un suelo.

La descomposición de la materia orgánica es un proceso biológico que ocurre naturalmente, por efecto de diferentes grupos de organismos, entre los cuales destacan los microorganismos del suelo, que permiten transformar los diferentes residuos, en nutrientes básicos y energía. Siendo que una de las funciones MOS es actuar como depósito de carbono, permitiendo que exista en el suelo más del doble del carbono contenido en la atmósfera, por lo que los cambios que experimente está, pueden tener un impacto en el equilibrio global, y por lo tanto un efecto sobre las condiciones climáticas del planeta, al ser considerado el carbono que sale de esta como CO₂, el cual es uno de los gases que contribuyen al efecto invernadero del planeta y al cambio climático (IPCC, 2000).

En la actualidad, se reporta un incremento en el cambio climático por efecto del impacto antropogénico, el cual se observa en el aumento de la temperatura en diferentes regiones, y como efecto de retroalimentación la temperatura influye sobre el proceso de descomposición y por lo tanto en el almacenamiento de carbono en el suelo y en la vegetación (Andriulo, 2006), todo esto muestra que el suelo es un compartimiento complejo debido a todas las interrelaciones que existen en el mismo.

Por lo que la comprensión de los diferentes procesos y transformaciones que ocurren en el suelo, son de gran importancia, sin embargo la cantidad de información que puede ser recolectada, así como la variabilidad en la misma, constituyen un desafío en sí, por lo cual esto conlleva ha utilizar herramientas tecnológicas que permitan el manejo de la información obtenida, es por ello que existe una gran cantidad de modelos que simulan y describen los procesos de la descomposición de la MOS, y los cuales pueden o están siendo acoplados a modelos de pronósticos a nivel global, de igual manera estos modelos están evolucionando, en la búsqueda de una mejor comprensión de los diversos procesos físico, químicos y biológicos [Larionova et al., 2007; Lomander et al., 1998].

Existen una gran cantidad de modelos que estudian la MOS, muchos de los cuales están desarrollados sobre plataformas propias, lo cual nos permite el proceso de enseñanza-aprendizaje, así como, de investigación en el desarrollo de mejoras al mismo por otros investigadores, de igual manera existen herramientas donde se han desarrollado de manera práctica modelos, tales como por ejemplo el modelo MOMOS (Materia Organica y MicroOrganismos del Suelo), el cual ha sido calibrado y validado con datos e información de un estudio en un gradiente altitudinal tropical [Pansu et al., 2010, 2014] mostrando buenos resultados en las diferentes evaluaciones, este mismo modelo ha sido evaluado, encontrando una propuesta de evolución del mismos (Valery, 2018). Sin embargo, las investigaciones antes mencionadas, presenta la desventaja de haber sido desarrolladas en un software de simulación privativo llamado Vensim creado por Ventana Systems, Inc. para la construcción de modelos de sistemas dinámicos, limitado a funciones desarrolladas por dicha empresa, lo que no permite ampliar la investigación en otros aspectos de interés sin la necesidad de acudir a otros softwares externos.

De igual manera, el grupo de Investigación en Biotecnología Agrícola y Ambiental (GIBAA) de la UNET, ha desarrollado diversos planteamientos de modelos bajo un software libre, por lo que se hace uso de la herramienta para el análisis estadístico y gráfico llamado R, el cual es un ambiente de programación formado por un conjunto de herramientas muy flexibles que pueden ampliarse fácilmente mediante paquetes, librerías o definiendo funciones propias. Además, es gratuito y de código abierto. Cualquier usuario puede descargar y crear su código de manera personal, sin restricciones de uso, la única regla es que la distribución siempre sea libre (GPL). Gracias a que puede accederse libremente a su código, R no tiene limitadas sus funciones, al contrario de lo que sucede con otras herramientas comerciales.

La finalidad de esta investigación consiste en utilizar el modelo MOMOS desarrollado por Valery

(2018) en Vensim para migrarlo hacia R, permitiendo la comparación de los resultados obtenidos en Vensim (Valery, 2018) contra el resultado obtenido del paquete en R. Buscando como meta la versatilidad a los investigadores interesados de poder trabajar sobre un ambiente que puedan manejar y modificar con las diferentes necesidades de su trabajo.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un paquete en el lenguaje R implementando el modelo MOMOS.

1.2.2. Objetivos Específicos

- Estudiar la Estructura del modelo MOMOS.
- Crear las funciones primarias en el lenguaje R de la implementación del modelo MOMOS.
- Realizar las pruebas unitarias y funcionales del paquete creado en el lenguaje R.

1.3. Justificación e Importancia

En las ciencias agronómicas y áreas afines, en la última década, se han utilizado una serie de herramientas que permiten mejorar los procesos de producción, así como del manejo de los recursos, entre las mismas, se encuentran los modelos dinámicos de simulación los cuales lograrán mostrar una representación lo más cercana a la realidad, generando un mejor entendimiento de los procesos en los ecosistemas y agroecosistemas.

1.4. Alcance y Limitaciones

El paquete se desarrolla en el lenguaje R y permite simular el modelo MOMOS, el resultado obtenido se comparará con los obtenidos por Valery (2018) en el programa VENSIM para simular el mismo modelo.

Capítulo 2

Fundamentos Teóricos

2.1. Antecedentes

Contreras (2018) construyó: “DimBio, Paquete en lenguaje R para la discriminización de modelos de simulación dinámicos para procesos biológicos.”, en este trabajo se expone de forma detallada la realización de un paquete en lenguaje R, para la discriminación de modelos de simulación dinámicos para procesos biológicos, para unificar las estadísticos que evalúan el desempeño de los modelos dinámicos, mediante el cálculo para discriminación de estadísticos univariantes y multivariantes utilizados para encontrar un modelo adecuado o que mejor se ajustará al proceso de estudio basado en modelos.

La investigación fue de relevancia para este trabajo de grado, ya que en ella crearon un paquete en R, y utilizaron modelos de simulación dinámicos.

Darghan (2018) realizó: “CGR, Paquete en lenguaje R, para el cálculo de índices fisiológicos de crecimiento y componentes del rendimiento en plantas.” trabajo en el cual se expone el desarrollo del paquete CGR cuyo interés radicó en el cálculo de aquellos índices que dependen de la primera derivada de la forma funcional ajustada al modelo de crecimiento de las plantas, también permite el cálculo de índices instantáneos en cualquier punto de interés por el usuario dentro de la región experimental o dentro del dominio de la función ajustada.

La investigación citada fue tomada como referencia, puesto que realizó cálculos de índices para evaluar el comportamiento de ciertas variables en procesos biológicos, en este caso, el crecimiento. Lo cual está vinculado al contexto biológico de los datos que fueron evaluados por el paquete que

se desarrolló.

Ablan (2011) construyó: “Una librería en R para validación de modelos de simulación”, en este trabajo se manejaron métodos para la validación de modelos de simulación continua para evaluar la calidad del modelo comparando los resultados de series temporales de datos reales con los resultados o salidas obtenidas de una simulación para el mismo lapso temporal, en este se utilizaron los criterios o índices de validación que pueden ser usados para la comparación de datos y resultados de un modelo, los cuales fueron: el error cuadrático medio, el estadísticos de Theil, el error absoluto medio, el índice de acuerdo de Willmott, la eficiencia del modelo de Nash Sutcliffe, el coeficiente de correlación de Pearson, el coeficiente de determinación, la prueba F, la prueba t pareada, el estadístico Hotelling T2, el criterio de información de Akaike y la inferencia bayesiana. Con estos criterios o índices se creó una librería del programa estadístico R que permitió el uso de estos índices para la validación de modelos de simulación.

La investigación fue importante en este trabajo de grado, ya que en ella crearon una librería de R, y utilizaron métodos estadísticos para respuesta univariante que son utilizados para la comparación de datos reales contra datos de modelos de simulación.

2.2. Bases Teóricas

2.2.1. Modelos

En la actualidad el avance de la tecnología, permite que para el estudio de sistemas biológicos, los modelos de simulación sean una herramienta que ayudan en el manejo de la información, así como la evaluación y planificación de manejo y diseño de experimentos.

Existe una gran cantidad de modelos biológicos, los cuales están relacionados con diversos procesos en los ecosistemas o agroecosistemas. Entre estos se pueden observar modelos teóricos, diagramáticos o matemáticos. “Estos modelos han venido evolucionando desde la última década, buscando entender los procesos que permiten el almacenamiento o la pérdida del suelo.” (Manzoni, 2009).

El uso de modelos en el área ecológica puede estar enfocados en diversos objetivos, entre los que se pueden destacar: la representación de diversas variables de estudio y sus tasas de cambio;

la descripción de ecosistemas y los procesos temporales y/o espaciales de los mismos; evaluar las condiciones pasadas y predecir el comportamiento futuro de los ecosistemas en estudio; poner a prueba teorías o hipótesis sobre la estructura y el funcionamiento de los ecosistemas (Blanco, 2013).

Dentro del grupo de modelos que estudian las transformaciones de la materia orgánica del suelo (MOS), se presentan características similares entre ellos, tales como, estar conformados por compartimientos, que representan a los diferentes compuestos químicos o biológicos, también se pueden diferenciar, por el número de compartimientos que poseen, que pueden ir desde 2 hasta 200 compartimientos. Sin embargo, "debe tenerse en cuenta un equilibrio entre el detalle y la comprensión de la información, para no perder el sentido de la modelización y poder realizar simulaciones y predicciones acordes a la realidad" (Blagodatsky 1998; Bruun 2003; Kirschbaum 2002; Zhang 2008).

La literatura presenta el interés que existe por los modelos que tienen que ver con las transformaciones de la MOS, muestra de esto es la gran cantidad de modelos que son referenciados en el registro de modelos ecológicos (REM <http://ecobas.org>), donde se presenta la información de cerca de 684 modelos.

Según lo presentado por Haefner (2005), sobre la importancia de los modelos y su potencial para el estudio de sistemas naturales, se tiene que mediante los mismos se puede lograr:

1. Sintetizar, manejar y unificar una gran cantidad de información en forma de ecuaciones y sus relaciones, permitiendo entender y explicar el comportamiento de los sistemas biológicos.
2. Analizar el funcionamiento del sistema para predecir el comportamiento futuro, bajo las condiciones naturales o con intervención de manejo.

"Los modelos de simulación plantean un esquema básico de trabajo, el cual es fundamentado en tres etapas generales" (Haefner, 2005; Liu 2005), las cuales pueden ser descritas de la siguiente manera:

1.- Conceptualización: conocer en gran medida la realidad que se trata de modelizar, ser capaces de representar lógicamente y de manera conceptual el problema.

2.- Formalización: establecer correctamente las relaciones entre los elementos que conforman el sistema en estudio, de forma que sean comprensibles, lo que permite construir un diagrama de estas relaciones y posteriormente representarlo mediante un diagrama de *Forrester*.

3.- Evaluación: establecer la forma en que debe ser el procedimiento de resolución a emplear, y la manera de interpretarlo correctamente.

Las etapas de conceptualización y formalización se basan en toda la información de décadas pasadas, sin embargo, el proceso de evaluación presenta diversos problemas, entre los que destacan la recolección de información con características confiables que sea representativa de los sistemas y validar los resultados obtenidos por el modelo con los obtenidos experimentalmente mediante un análisis estadístico inferencial.

2.2.2. Lenguaje R.

R es un conjunto integrado de *software* de código abierto para el almacenamiento, manipulación, cálculo y visualización de datos para computación y graficación estadística, puede ser compilado y ejecutado en Windows, Mac OS X y otras plataformas UNIX (como Linux), se distribuye usualmente en formato binario (<https://www.r-project.org/about.html>, 2018). El proyecto de *software* R fue iniciado por Robert Gentleman y Ross Ihaka. El lenguaje fue influenciado por lenguaje S desarrollado originalmente en Bell Laboratories por John Chambers y sus colegas. Desde entonces ha evolucionado para el cálculo estadístico asociado a diversas disciplinas para contextos académicos y comerciales.

En R, la unidad fundamental de código compartible es el paquete, el cual agrupa código, datos, documentación y pruebas, y resulta simple de compartir con otros. Para enero del 2015 ya habían más de 6.000 paquetes disponibles en la Red Integral de Archivos de R, conocido comúnmente por su acrónimo CRAN, el cual es el repositorio de paquetes. Esta gran variedad de paquetes es una de las razones por las cuales R es tan exitoso, pues es probable que algún investigador o académico, ya haya resuelto un problema en su propio campo usando esta herramienta, por lo que otros usuarios simplemente podrán recurrir a ella para su uso directo o para llamarla en un nuevo código (Wickham, 2015). el desarrollo del algoritmo del modelo MOMOS se realiza empleando el lenguaje R y la herramienta Rstudio, así como también diversos paquetes disponibles.

2.2.3. RStudio.

RStudio es un ambiente de desarrollo integrado (*Integrated Development Environment*, IDE) que ofrece herramientas de desarrollo vía consola, editor de sintaxis que apoya la ejecución de

código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo. RStudio está disponible para Windows, Mac y Linux o para navegadores conectados a RStudio Server o RStudio Server Pro (Debian / Ubuntu, RedHat / CentOS, y SUSE Linux) (<https://www.rstudio.com/about/>, 2018). También es necesario conocer con precisión que tipo de lenguaje es y una buena forma de realizar el algoritmo que simula el modelo MOMOS.

2.2.4. Algoritmos en R/RStudio.

El lenguaje R es un lenguaje interpretativo y no compilado, esto se traduce a que los comandos escritos por teclado son ejecutados directamente sin necesidad de construir un ejecutable, lo cual representa una facilidad al momento de analizar datos complejos.

Como lenguaje de programación, R adopta una sintaxis y una gramática que difiere de muchos otros lenguajes: los objetos en R son vectores, y las funciones son vectorizadas para operar con todos los elementos del objeto. Los objetos en R tienen una semántica de copia sobre el cambio y de paso por valor, reduciendo las consecuencias inesperadas para los usuarios en detrimento del uso menos eficiente de la memoria. Es importante puntualizar que los paradigmas comunes en otros lenguajes, como el bucle 'for', no son tan usados en R. Los programas escritos en R reflejan de una manera más clara los algoritmos estadísticos que implementan. Además, cuenta con la ventaja de que es fácil de depurar código en R por ser un lenguaje interpretado. (Becerro, 2014).

Para poder comparar el modelo MOMOS experimental versus el modelo simulado, es necesario leer los datos experimentales, como método de obtención se emplea la lectura de datos desde un archivo de formato ".xls". Los datos leídos son aquellos datos experimentales del modelo que fueron capturados en un momento determinado y que escapa del contexto de este estudio.

xlsx: es un paquete para R que da control programático de archivos excel. Este paquete emplea una API de alto nivel que permite al usuario leer de una hoja de un archivo .xlsx a un data.frame y escribir un data.frame a un archivo. Funcionalidad de menor nivel permite la manipulación directa de hojas, filas y celdas.

ggplot2: es un paquete para R que permite crear elegantes visualizaciones de datos usando gramática de descripción de gráficos, es un sistema para declarativamente crear gráficos. Una vez la data ha sido generada, mediante funciones se le indica a este paquete como graficarlas.

2.2.5. Estructura de paquetes en R/RStudio.

La estructura de un paquete en R cuenta con al menos los cuatro ítems siguientes:

1. **DESCRIPTION:** En este componente se encuentra la metadata del paquete. La tarea del archivo Description es de gran importancia, ya que es en donde se registra la metadata, las dependencias que utiliza el paquete, la licencia y el soporte en caso de ocurrir errores con el mismo.

La estructura mínima para realizar un DESCRIPTION de un paquete en R es la siguiente:

- Package: mypackage
 - Title: What The Package Does (one line, title case required)
 - Version: 0.1
 - Authors@R: person("First", "Last", email = "first.last@example.com",
 - role = c("aut", "cre"))
 - Description: What the package does (one paragraph)
 - Depends: R (*i*= 3.1.0)
 - License: What license is it under?
 - LazyData: true
2. **R/Directorio:** Dirección del repositorio donde se encuentra el código del paquete, es decir los ".R" archivos. Se exponen las buenas prácticas a la hora de realizar todo el código en R, desde organización de las funciones, estilos de código, comentarios y nombre de variables.
 - Organizar funciones en R: Aunque se puede organizar los archivos como se desee, los dos extremos son malos, no colocar todas las funciones en el mismo archivo y no crear un archivo para cada función, aunque si una función es muy grande o tiene mucha documentación se puede dar el caso. Los nombres de los archivos tienen que tener un significado y deben de terminar en R. Se puede recomendar de acuerdo al número de funciones utilizar prefijo.

- Nombres de objetos: Los nombres de las variables y funciones deben de ser en minúsculas, usar el guión bajo (_) para separar palabras. En lo posible no debe usarse nombres de variables existentes, esto causará confusión.
- Comentarios: Para comentar el código, se comienza con #, los comentarios deben de explicar el porqué, no el que. Se usan los caracteres (-) y (=) para separar líneas.
- Estilos de código: Existe diferencia entre el código utilizado en *scripts* y paquetes: en un *script*, el código se ejecuta cuando se carga, mientras en un paquete el código se ejecuta cuando se genera. Esto significa que el código de paquetes solo debe crear objetos, a continuación se amplían estas importantes diferencias:
 - Cargando código: Cuando se carga un *script* el código se ejecuta de una vez, el procedimiento es diferente en un paquete, porque es cargado en dos pasos, el primero, cuando se construye el paquete todo el código se ejecuta en R/ y su resultado es guardado, el otro paso es cuando se carga un paquete, con *library()* o *require()*, los resultados almacenados en la caché están dispuestos para su uso.
 - The R landscape: Hay también diferencia en un *script* y un paquete, por lo tanto hay que prestar atención a R *landscape*, incluyendo las funciones, objetos y todas las variables globales.

Recomendaciones:

- No se suele utilizar *library()* o *require()*. Estos modifican la ruta de búsqueda, lo que afecta las funciones disponibles del entorno global. Es mejor usar la descripción para especificar los requisitos de un paquete.
- Nunca emplear *source()* para cargar el código de un archivo. Ya que *source()* modifica el entorno actual, insertando los resultados de ejecutar el código. En cambio, usando el *devtools::load_all()* automáticamente se generarán todos los archivos en R.

3. **Man/Documentación:** La documentación es uno de los aspectos más importantes de un buen paquete. Sin ella, los usuarios no sabrán cómo usar un paquete y además ayudan a recordar que realizan sus funciones para el futuro.

R proporciona una forma estándar de documentar los objetos en un paquete: Se escriben archivos .Rd en el directorio “man”. Éstos utilizan LaTeX, y se procesan en HTML, texto plano y pdf para su visualización. En lugar de escribir estos archivos a mano, se emplea “roxygen2”, que convierte los comentarios especialmente formateados en archivos “.Rd”, el objetivo de roxygen2 es hacer que documentar el código sea lo más fácil posible.

Los comentarios roxygen utilizan `#` y se utilizan para distinguir de los comentarios regulares.

Para documentar funciones se utilizan generalmente 3 etiquetas que son:

- `@param`: donde se describe los parámetros de entrada de la función.
- `@examples`: donde se muestra un ejemplo funcional de la función ya que R CMD chequea que sea correcto.
- `@return`: especifica lo que retorna la función siempre que sea necesario.

Para documentar el paquete se recomienda utilizar roxygen con la etiquetas `@docType package` y `@name` (nombre del paquete) también se puede utilizar la etiqueta `@section` que permite ser específicos al momento de documentar.

Documentación de clases, genéricos y métodos:

- **S3**: La clase S3 no tiene definición formal así que documenta la función constructora. No se necesita documentar métodos simples pero si el método es complicado o incluye argumentos adicionales se debe de documentar.
- **S4**: La clase S4 utiliza roxygen para realizar la documentación, los S4 también son funciones así que se deben documentar como tal, la diferencia con S3 es que todos los métodos deben documentarse. A menudo el código S4 necesita ejecutarse en cierto orden, por ejemplo, para definir los métodos `setMethod("foo", c("bar", "baz"), ...)` se deben haber creado antes el `"foo"` genérico y las 2 clases.
- **RC**: Las clases RC de referencia son diferentes a S3 y S4 porque los métodos están asociados con clases, no con genéricos. La `"docstring"` es una cadena colocada dentro de la definición del método que describe brevemente lo que hace. Esto hace que documentar RC sea más simple que S4 porque solo se necesita un bloque de roxygen por clase.

4. **NAMESPACE**: Especifica que objetos conforman el paquete. Se utiliza para proporcionar espacios a los nombres como su nombre lo indica, es utilizado para interactuar con los paquetes y sus variables del CRAN de R.

2.2.6. Pruebas estándar y pruebas funcionales

Para poder evaluar si el algoritmo y su componentes satisfacen o no los requerimientos para el modelo MOMOS, es necesario realizar un proceso de pruebas. Una prueba es definida según

Tutorialspoint (Software testing, sf.) como *“A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item”*.

En el contexto de este estudio se consideran las pruebas estándar como el conjunto de pruebas de nivel que comprenden las pruebas unitarias, las pruebas de sistema y las pruebas de aceptación.

- **Pruebas Unitarias:** “Unit testing is performed by the respective developers on the individual units of source code assigned areas. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality”. La data para efectuar estas pruebas suele ser diferente de la que es empleada en las pruebas de aceptación.
- **Pruebas de Sistemas:** “System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards”, (Tutorialspoint, Software testing, sf). Las pruebas de sistema permiten probar verificar y validar los requerimientos del sistema en conjunto con la arquitectura la aplicación.
- **Pruebas de Aceptación:** “Acceptance tests are (...) to point out simple spelling mistakes, cosmetic errors, or interface gaps, (and) to point out any bugs in the application that will result in system crashes or major errors in the application”, (Tutorialspoint, Software testing, sf). Estas pruebas son para probar como la aplicación se comporta en producción.

Existe un tipo de prueba especial que es la que se va aplicar al final del desarrollo del algoritmo, estas son las pruebas funcionales. Tutorialspoint (Software testing, sf.) define las pruebas funcionales como:

“This is a type of (...) testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system’s compliance with its specified requirements”.

TODO: Definir terminos

2.2.7. Glosario

Código fuente: Conjunto de instrucciones que componen un programa, escrito en cualquier lenguaje. En inglés "source code".

Crecimiento: Aumento en un atributo medido, x , de un órgano u organismo, como una función del tiempo, t ; $x = f(t)$.

Data: El nombre genérico para cualquier cosa que entre, salga o se guarde en una computadora o cualquier otro medio, siempre y cuando sea todo en formato digital.

Modelo: «Un modelo (M) para un sistema (S) y un experimento (E) es cualquier objeto al que puede aplicarse E para responder preguntas acerca de S» Marvin Minski (1965). Paul Humphrey define simulación como "Simulación es cualquier método implementado en computadora destinado a explorar las propiedades de modelos matemáticos, cuando no se dispone de métodos analíticos" (Grüne-Yanoff y Weirich 2010).

Lenguaje R: R es un entorno y lenguaje de programación con un enfoque al análisis estadístico.

R Studio: R Studio es un entorno de desarrollo integrado, en inglés "integrated development environment" (IDE) para R.

Necromasa:

HL:

HLo:

HS:

HSo:

Co:

Kvl:

Kvs:

fs:

Khl:

Khs:

Khls:

Kmb:

Kresp:

Ci:

VL:

VS:

CM:

RA:

Fvl:

Fvs:

Fmor:

Resp:

Fhl:

Fhls:

Fhs:

dVL:

dVS:

dCM:

dHL:

dHS:

dRA:

2.3. *GNU General Public License v2.0 (GPL-2.0)*

Es una licencia de *software* libre, que garantiza que se pueda copiar, distribuir y modificar el *software* siempre que realice un seguimiento de los cambios / fechas en los archivos de origen. Cualquier modificación o software que incluya (a través del compilador) el código con licencia GPL también debe estar disponible bajo la GPL junto con las instrucciones de compilación e instalación.

Capítulo 3

Fundamentos Metodológicos

A continuación se plantea la metodología para el presente trabajo, detallando el enfoque, tipo, nivel y diseño de la investigación y la metodología que se implementa.

3.1. Enfoque de la investigación

La presente investigación se desarrolla siguiendo un enfoque cuantitativo, puesto que, como lo indican Pallela y Martins (2012), “la investigación cuantitativa requiere el uso de instrumentos de medición y comparación, que proporcionan datos cuyo estudio necesita la aplicación de modelos matemáticos y estadísticos, el conocimiento está basado en hechos”. Los datos usados en el desarrollo del paquete y para la comparación de los resultados, provienen de la tesis Doctoral de Valery (2018).

3.2. Tipo o nivel de investigación

Este proyecto planteó un tipo de investigación de desarrollo, en donde se integra la programación y evaluación del comportamiento de un paquete de simulación, que permite indagar los efectos de la interrelación entre los diferentes tipos de variable en lugar de los hechos.

En este punto se determinó la profundidad que abarca esta investigación, teniendo en cuenta que de acuerdo con el nivel de la investigación es definido como “grado de profundidad con que se aborda un fenómeno u objeto de estudio” (Arias, 2012).

En este sentido, se tiene que dadas las características del proyecto, se asocia con un nivel descriptivo, tal como lo indican Pallela y Martins (2012), “hace énfasis sobre conclusiones dominantes o sobre como una persona, grupo o cosa se conduce o funciona en el presente” esto debido a que se midieron los datos extraídos sin alterarlos para ser mostrados en el sistema.

3.3. Diseño de la investigación

Según Arias(2012), el diseño de la investigación es “la estrategia general que adopta el investigador para responder al problema planteado” (p.21) por lo que es vital haber establecido una correcta secuencia de pasos para la elaboración del prototipo de software que dio solución a la problemática principal de la investigación.

Con este enfoque, se tiene que este trabajo siguió un diseño no experimental, enfocado en el uso de información existente, de acuerdo con lo dicho por Pallela y Martins (2012) al definir el diseño no experimental como:

Es el que se realiza sin manipular en forma deliberada ninguna variable. El investigador no sustituye intencionalmente las variables independientes. Se observan los hechos tal y como se presentan en su contexto real y en un tiempo determinado o no, para luego analizarlos. Por lo tanto, este diseño no se construye una situación específica sino que se observan las que existen. Las variables independientes ya han ocurrido y no pueden ser manipuladas, lo que impide influir sobre ellas para modificarlas. (p.81)”

Esto indica que no hay manipulación de variables. Esta investigación presenta una modalidad de proyecto especial que, como lo indican Pallela y Martins (2012), los proyectos especiales “destinados a la creación de productos que puedan solucionar deficiencias evidenciadas, se caracterizan por su valor innovador y aporte significativo” (p.92), ya que se creó un *software* aplicable al área de estudio.

3.4. Metodología

Para el desarrollo del paquete se siguió las pautas estándar establecidas para la creación de paquetes y extensiones en R.

Creación del esqueleto del paquete.

En esta etapa se diseñaron y crearon los directorios, ficheros y objetos que conforman el paquete.

Registrar el método para el envío y uso de funciones.

En esta etapa del desarrollo se estableció las dependencias sobre los paquetes de la base fuente de código R y sus métodos de conexión, considerando el manejo de versiones y los criterios de mantenimiento, además se estableció los espacios de nombre o las estrategias para la búsqueda y utilización de las variables; unificando estos criterios a las funciones que fueron diseñadas.

Diseño y codificación de las funciones.

Los métodos para el diseño de las funciones primarias en R fueron los diagramas de flujo; y para su codificación se siguieron las normas de estilo para codificación en R, sugeridas por Wickham (2015) y por el creador del paquete *formatR* Xie(2017), además se estableció la dependencia con las funciones de código base y las recomendadas para desarrollo en R.

Pruebas unitarias de las funciones.

Debido a que los paquetes en R están conformados, entre otros elementos por las funciones primarias, a cada una de ellas se les realizaron pruebas unitarias en dos fases, la primera con datos sintéticos que permiten comprobar cada estado del diagrama de flujo, que esquematiza la solución numérica que permite el cálculo, para esto se utilizó el paquete de *RUnit* (Zenka, 2015) y *testthat* (Wickham, 2017), y la segunda etapa donde el modelo final ya implementado permite realizar pruebas con los conjuntos de datos facilitados que formaron parte integral del paquete y con los cuales se desarrollaron los ejemplos prácticos que conformaron la documentación que acompaña al paquete R.

Chequear la carga del paquete.

En esta etapa del desarrollo se utilizó las funciones de chequear el paquete que ofrece el código R; cuya finalidad es verificar cada fichero del árbol de carpetas asociadas a cada elemento de la estructura o esqueleto del paquete, que a su vez crea el archivo de documentación en LaTeX y/o HTML, compila el código fuente y crea las librerías de enlace dinámico (*dynamic link library* DLL).

Construcción del método de distribución del paquete.

Se seleccionó la forma de distribución del paquete desde el repositorio local, creando los ficheros fuentes (en formato *tarball*) y en binario.

3.5. Aspectos administrativos

La realización de la investigación fue planificada según lo establecido en el siguiente diagrama:

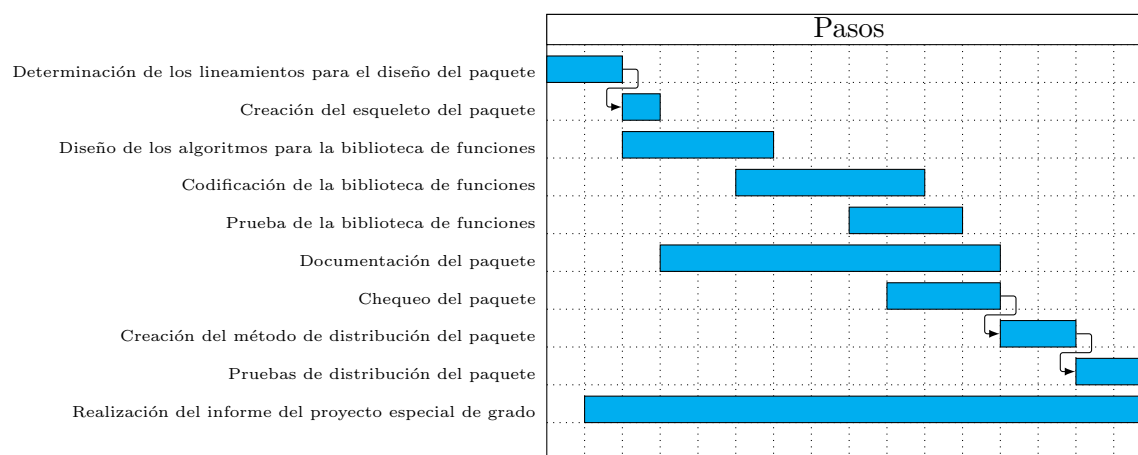


Figura 3.1: Diagrama de Gantt con la planificación del proyecto especial de grado

Capítulo 4

Desarrollo

A continuación se describe de forma detallada, el diseño y desarrollo del paquete MOMOS (Materia Organica y MicroOrganismos del Suelo), siguiendo la metodología descrita en el capítulo 3.

4.1. Creación del esqueleto del paquete

Para esta fase se utiliza el ambiente de desarrollo RStudio, el cual es un entorno de desarrollo integrado de fuente abierta para R que agrega muchas características y herramientas de productividad, además de facilitar el uso de R, integrando la ayuda y la documentación.

En esta primera etapa se crea la estructura del paquete, además, se cargan los paquetes necesarios del CRAN de R (Repositorios de R), de los cuales depende MOMOS; los mismos fueron almacenados en el repositorio local que se crea para este paquete durante el proceso de desarrollo, y para el manejo de versiones se utiliza Git y GitHub, los principales paquete empleados son:

- devtools(Wickham, H. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/devtools/index.html>)
- roxygen2(Wickham, H. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/roxygen2/index.html>)
- testthat(Wickham, H. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/testthat/index.html>)

- RUnit(Burger, M. *et. al*, 2018; disponible en <https://cran.rstudio.com/web/packages/RUnit/index.html>)
- ggplot2(Wickham, H. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/ggplot2/index.html>)
- reshape2(Wickham, H. *et. al*, 2020; disponible en <https://cran.r-project.org/web/packages/reshape2/index.html>)
- deSolve(Soetaert, K. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/deSolve/index.html>)
- minpack.lm(Elzhov, T. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/minpack.lm/index.html>)
- xlsx(Dragulescu A. *et. al*, 2020; disponible en <https://cran.r-project.org/web/packages/xlsx/index.html>)
- FME(Soetaert K. *et. al*, 2021; disponible en <https://cran.r-project.org/web/packages/FME/index.html>)
- knitr(Xie Y. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/knitr/index.html>)
- rmarkdown(Allaire JJ. *et. al*, 2022; disponible en <https://cran.r-project.org/web/packages/rmarkdown/index.html>)

Los pasos para la creación del esqueleto del paquete son los siguientes:

- Como se muestra en la Figura 4.1, dentro del entorno R Studio, en el menú principal, se selecciona la opción "File" y en el submenú desplegado se selecciona "New Project..."

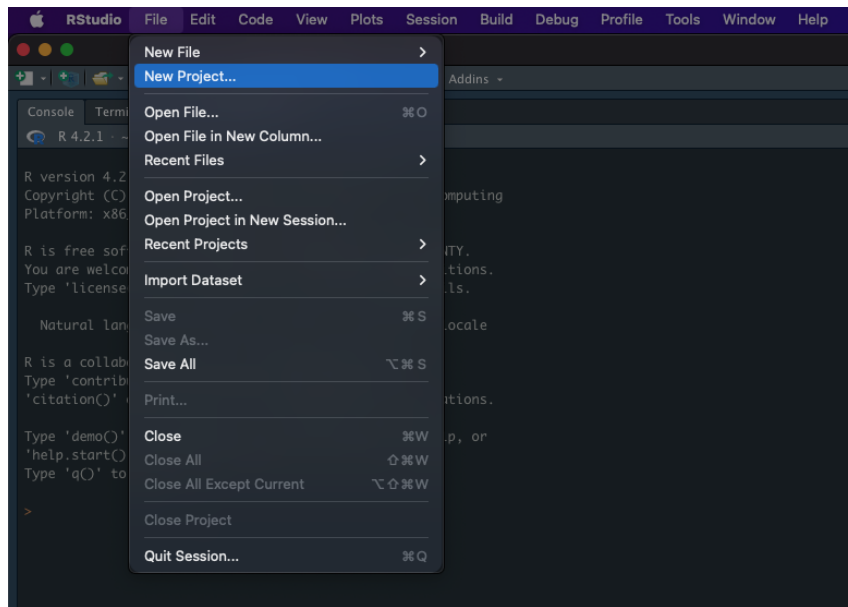


Figura 4.1: Nuevo proyecto

- En la Figura 4.2, se selecciona la opción *"New Directory"*

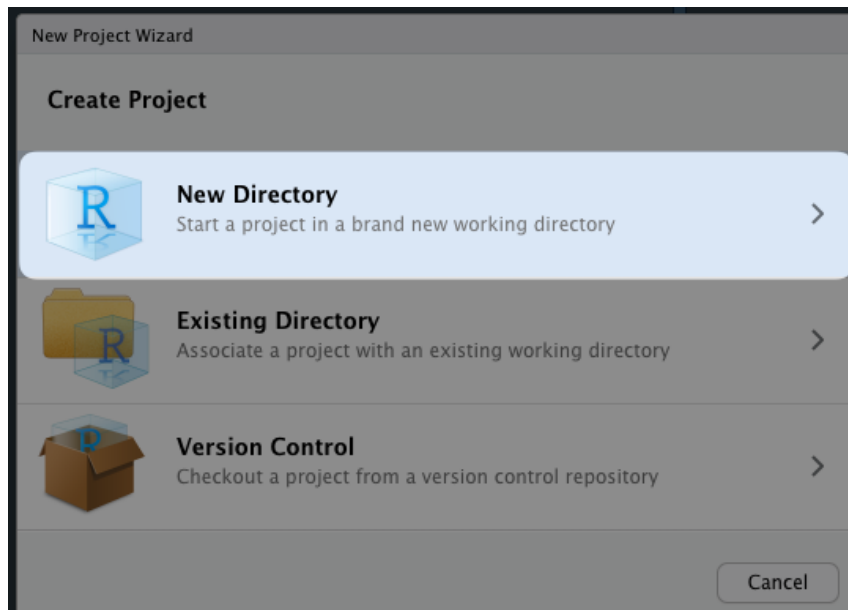


Figura 4.2: Nuevo directorio

- En la figura 4.3, se selecciona la opción *"R Package"*

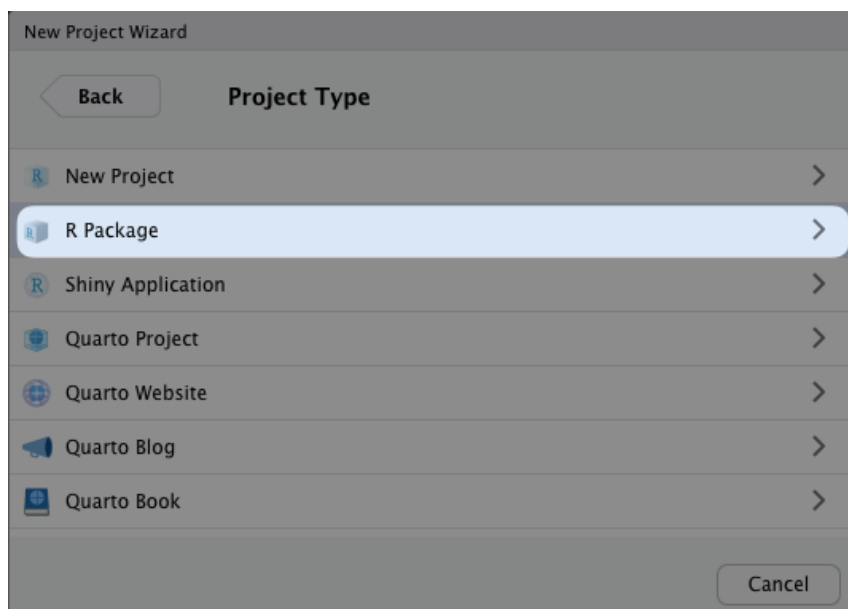


Figura 4.3: Tipo de proyecto

- En la figura 4.4, se asigna el nombre del paquete y se selecciona el directorio donde es trabajado, posteriormente se presiona el botón *"Create Project"*

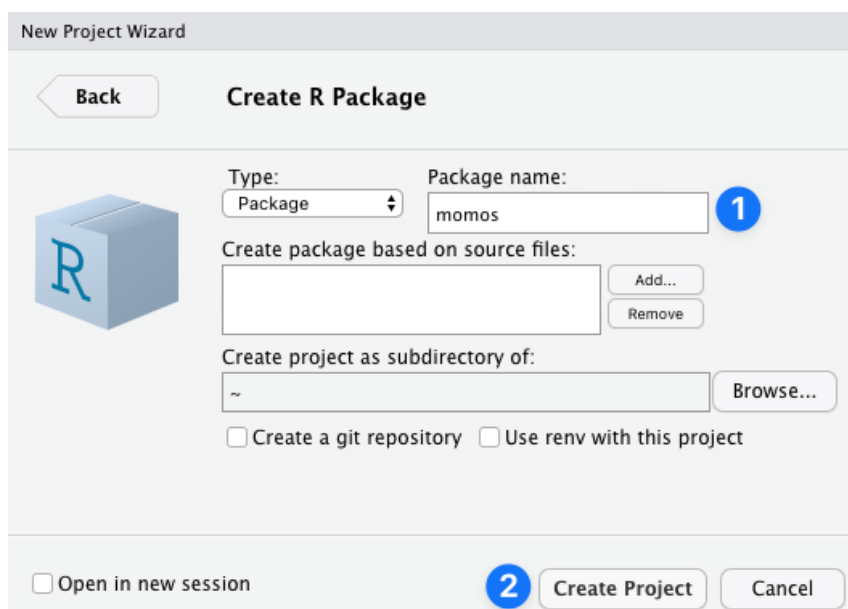
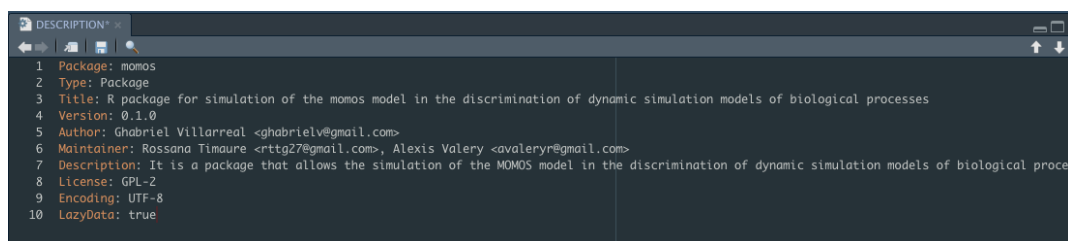


Figura 4.4: Nombre del paquete y directorio de trabajo

- En la figura 4.5, se observa la descripción del paquete



```

1 Package: momos
2 Type: Package
3 Title: R package for simulation of the momos model in the discrimination of dynamic simulation models of biological processes
4 Version: 0.1.0
5 Author: Ghabriel Villarreal <ghabriely@gmail.com>
6 Maintainer: Rossana Timaure <rttg27@gmail.com>, Alexis Valery <avalery@gmail.com>
7 Description: It is a package that allows the simulation of the MOMOS model in the discrimination of dynamic simulation models of biological processes
8 License: GPL-2
9 Encoding: UTF-8
10 LazyData: true

```

Figura 4.5: Estructura archivo DESCRIPTION

- En la figura 4.6, finalmente se observa el esqueleto del paquete



File/Folder	Size
..	
.Rbuildignore	28 B
DESCRIPTION	367 B
man	
momos.Rproj	356 B
NAMESPACE	31 B
R	

Figura 4.6: Esqueleto del paquete

4.2. Registrar el método para el envío y uso de funciones

Se determina que la función principal reciba una lista de parámetros de entrada que sirven para calcular los valores de simulación del modelo, estos pueden ser opcionales, y todos deben ser numéricos, en caso de que no se defina algún parámetro se cargan los valores por defecto correspondientes.

TODO: Agregar lista de parametros por defecto (tabla)

La primera función es *calculate_momos()* cuya salida muestra los valores de CM y RA simulados por cada instante de tiempo en un intervalo determinado, esta salida Figura 4.7 es un *data frame* que corresponde a los datos simulados por el modelo dinámico. Entendiéndose como un data frame según Santana y Nieves (2014) “una clase de objetos especial en R. Normalmente, cuando se realiza un estudio estadístico sobre los sujetos u objetos de una muestra, la información se organiza en un dataframe: una hoja de datos, en los que cada fila corresponde a un sujeto y cada columna a una variable. La estructura de un data frame es muy similar a la de una matriz. La diferencia es que una matriz sólo admite valores numéricos, mientras que en un data frame podemos incluir

también datos alfanuméricos.”

Por lo tanto, para esta función desarrollada, las columnas del data frame corresponden a las variables de estudio (CM / RA) (TODO: Agregar definicion de CM / RA) y las filas a los valores de las mismas en cada intervalo de tiempo.

El modelo dinámico genera una salida sin calibrar, esto puede producir que al comparar los datos reales con los datos simulados exista una gran diferencia entre los datos, por esta razón, se decide diseñar una segunda función *calibrate_momos()* para realizar la calibración de los parámetros de entrada que afectan directamente las variables de salida (CM / RA), esto con el fin de buscar los valores ideales usando el algoritmo de levenberg marquart (TODO: Agregar informacion del algoritmo), con estos nuevos valores Figura 4.8 se busca asegurar un margen de error menor.

Adicionalmente, el resultado de esta función muestra también los valores ideales de los parámetros que fueron usados para calibrar el modelo iterando nuevamente sobre la función *calcular_momos()*.

Para representar los valores obtenidos de las funciones anteriormente indicadas se define la función *graph_momos()* la cual tiene como finalidad comparar de manera grafica la curva del modelo con los valores simulados iniciales, los valores reales y también los nuevos valores simulados después de la calibración.

Por último, para unificar todas las funciones y mejorar el uso del paquete se crea la función *momos()* que contempla todas las salidas de dichas funciones.

```
> library(momos)
> momos()
      CM      RA
1  50.04000  0.0000
2 347.89934 245.6057
3 368.49484 668.1389
4 360.32194 1038.2973
5 337.31676 1357.5673
6 310.19164 1627.7234
7 283.82091 1854.0023
8 259.64030 2043.1562
9 237.86573 2201.6005
10 218.37927 2334.8244
11 200.99087 2447.3685
12 185.50634 2542.9515
13 171.74363 2624.6102
14 159.53579 2694.8246
15 148.73058 2755.6206
16 139.18931 2808.6549
17 130.78552 2855.2839
18 123.40378 2896.6196
19 116.93858 2933.5750
20 111.29344 2966.9007
21 106.38005 2997.2153
22 102.11767 3025.0290
23  98.43248 3050.7640
24  95.25716 3074.7696
25  92.53046 3097.3357
26  90.19685 3118.7033
27  88.20615 3139.0727
28  86.51329 3158.6107
29  85.07791 3177.4564
30  83.86417 3195.7257
> |
```

Figura 4.7: Datos de salida sin calibrar

```
> calibrate_momos()
      CM      RA
1  50.04000  0.0000
2  95.99952 497.5417
3 127.02085 909.8104
4 158.61259 1240.3441
5 180.25281 1515.0821
6 190.50106 1747.9557
7 188.52944 1949.9086
8 178.35583 2125.1159
9 165.21227 2274.9809
10 152.07010 2401.9059
11 139.95188 2509.2205
12 129.06790 2600.2396
13 119.38400 2677.8530
14 110.80708 2744.4670
15 103.23571 2802.0574
16  96.57274 2852.2396
17  90.72794 2896.3341
18  85.61812 2935.4209
19  81.16674 2970.3841
20  77.30339 3001.9486
21  73.96328 3030.7095
22  71.08690 3057.1562
23  68.61969 3081.6914
24  66.51178 3104.6471
25  64.71781 3126.2977
26  63.19669 3146.8695
27  61.91146 3166.5498
28  60.82908 3185.4934
29  59.92022 3203.8284
30  59.15902 3221.6608
> |
```

Figura 4.8: Datos de salida calibrados

4.3. Diseño y codificación de las funciones

La codificación de las funciones se realizaron en el IDE(Entorno de desarrollo integrado) RStudio, siguiendo las especificaciones algorítmicas detalladas en sus respectivos diagramas de flujo.

Diseño de las soluciones algorítmicas, para el cálculo del modelo momos

El objetivo funcional de *calculate_momos()* es obtener los valores resultantes de la ejecución de MOMOS planteado por Valery (2018) para ello es necesario establecer los valores por defecto de los parámetros y constantes facilitados por el autor del modelo de esta investigación o los definidos por el usuario del paquete:

```
# Initial values
Necromasa <- ifelse("Necromasa" %in% names(params), params$Necromasa, 2140)
HLo <- ifelse("HLo" %in% names(params), params$HLo, 2250)
HSo <- ifelse("HSo" %in% names(params), params$HSo, 19150)
Co <- ifelse("Co" %in% names(params), params$Co, 55.40)

# Parameters of the MOMOS model
Kvl <- ifelse("Kvl" %in% names(params), params$Kvl, 0.2070)
Kvs <- ifelse("Kvs" %in% names(params), params$Kvs, 0.00057)
fs <- ifelse("fs" %in% names(params), params$fs, 0.00002)
Khl <- ifelse("Khl" %in% names(params), params$Khl, 0.0638)
Khs <- ifelse("Khs" %in% names(params), params$Khs, 0.00077)
Khls <- ifelse("Khls" %in% names(params), params$Khls, 0.1581)
Kmb <- ifelse("Kmb" %in% names(params), params$Kmb, 0.001)
Kresp <- ifelse("Kresp" %in% names(params), params$Kresp, 0.1419)
Ci <- ifelse("Ci" %in% names(params), params$Ci, 50.04)

# Execution time
from <- ifelse("from" %in% names(params), params$from, 1)
to <- ifelse("to" %in% names(params), params$to, 30)
at <- ifelse("at" %in% names(params), params$at, 1)
```

Figura 4.9: Establecer parámetros y constantes

Los valores de los parámetros de tiempo de ejecución (from, to, at) indican la cantidad de veces y de datos que se debe iterar MOMOS. Adicionalmente, es necesario calcular los compartimientos de la necromasa labil (VL), necromasa estable (VS), carbono inicial de la biomasa microbiana (CM), humus labil (HL), humus estable (HS) y respiración inicial de la biomasa microbiana (RA) para obtener la función inicial.

```
y <- c(
  VL = Necromasa * (1 - fs),
  VS = Necromasa * fs,
  CM = Ci,
  HL = HLo,
  HS = HSo,
  RA = 0
)
```

Figura 4.10: Establecer la función inicial y

Posteriormente, se calculan las N derivadas de las funciones de los compartimientos anteriormente descritos, donde N es rango de cantidad de derivadas que se deben aplicar según los parámetros de tiempo de ejecución.

```
# Derivative
derivs <- function(times, y, params) {
  with (as.list(c(params, y)), {
    FvI <- VL * KvI
    FvS <- VS * KvS
    FmR <- CM * Kmb
    Resp <- CM * CM * Kresp / Co
    FhI <- HL * KhI
    FhLS <- HL * KhLS
    FhS <- HS * KhS

    dVL <- -FvI
    dVS <- -FvS
    dCM <- FvI + FvS - FmR - Resp + FhI + FhS
    dHL <- FmR - FhI - FhLS
    dHS <- FhLS - FhS
    dRA <- Resp

    return (list(c(dVL, dVS, dCM, dHL, dHS, dRA)))
  })
}
```

Figura 4.11: Función derivación

Una vez se obtienen las N derivadas necesarias, se procede a utilizar la librería *deSolve* haciendo uso de la función *ode()* que resuelve un sistema de ecuaciones diferenciales ordinarias, dicha función recibe los siguientes parámetros:

- *func*: una función que calcula los valores de las derivadas en el sistema ODE (la definición del modelo) en el tiempo t .
- *y*: estado inicial del sistema ODE.
- *params*: parámetros que son recibidos por la función.
- *times*: secuencia de tiempo para la salida deseada, el primer valor es el inicial.

- *method*: método de integración usado, en este caso usamos *rk4* que el método de *Runge-Kutta*.

```
# Differential Equations
out <- ode(
  func = derivs,
  y = y,
  parms = pars,
  times = times,
  method = "rk4"
)
```

Figura 4.12: Ecuación diferencial

Como resultado de la ecuación diferencial se obtienen los datos simulados por el modelo MO-MOS en base a los parámetros y constantes establecidos, los cuales serán útiles para calibrar y luego comparar en las funciones posteriores, entre los datos simulados, datos calibrados y datos experimentales.

```

> out
      CM      RA
[1,] 50.04000  0.0000
[2,] 95.99952 497.5417
[3,] 127.02085 909.8104
[4,] 158.61259 1240.3441
[5,] 180.25281 1515.0821
[6,] 190.50106 1747.9557
[7,] 188.52944 1949.9086
[8,] 178.35583 2125.1159
[9,] 165.21227 2274.9809
[10,] 152.07010 2401.9059
[11,] 139.95188 2509.2205
[12,] 129.06790 2600.2396
[13,] 119.38400 2677.8530
[14,] 110.80708 2744.4670
[15,] 103.23571 2802.0574
[16,] 96.57274 2852.2396
[17,] 90.72794 2896.3341
[18,] 85.61812 2935.4209
[19,] 81.16674 2970.3841
[20,] 77.30339 3001.9486
[21,] 73.96328 3030.7095
[22,] 71.08690 3057.1562
[23,] 68.61969 3081.6914
[24,] 66.51178 3104.6471
[25,] 64.71781 3126.2977
[26,] 63.19669 3146.8695
[27,] 61.91146 3166.5498
[28,] 60.82908 3185.4934
[29,] 59.92022 3203.8284
[30,] 59.15902 3221.6608

```

Figura 4.13: Resultado de la ecuación diferencial

Diseño de las soluciones algorítmicas, para la calibración del modelo momos

El objetivo funcional de *calibrate_momos()* es obtener al menos un parámetro que pueda ser calibrado para encontrar su valor ideal, el cual permita que los datos simulados se acerquen a los datos experimentales, logrando así un margen de error que sea tolerante para la investigación.

En este caso se implementa el algoritmo de *levenberg-marquart* usando la librería de R *minpack.lm* para resolver problemas de mínimos cuadrados no lineales donde este algoritmo parte de los datos experimentales, por esta razón procedemos a leer los datos dentro de la función de cali-

bración.

```
# Getting experimental data
experimental_data <- read.xlsx(paste(getwd(),"data/momos.xlsx", sep="/"), sheetIndex = 1)
names(experimental_data)=c("time","CM_experimental","RA_experimental")
```

Figura 4.14: Extracción de datos experimentales

Para usar el algoritmo mencionado se debe crear un vector de residuos cuya suma cuadrada debe ser minimizada, y el primer argumento debe ser par, así que procedemos a crear una función que retorna ese vector, inicialmente, se debe hacer una unión de los tiempos de ejecución entre los datos experimentales y simulados, donde el tiempo debe ser único. Por otra parte, se establecen los parámetros de estimación y se procede a calcular MOMOS para el valor indicado, luego se evalúan los valores estimados versus los valores experimentales para los tiempos donde ambos coinciden aplicando una intersección entre los datos, finalmente, se restan los valores estimados con los valores experimentales, y se devuelve el vector a la función de calibración.

```
ssq=function(parms){
  # Time points for which values is experimental
  # Include the points where data is available
  t=c(seq(from,to,at),experimental_data$time)
  t=sort(unique(t))

  # parameters from the parameter estimation routine
  k1=parms[1]

  # solve the equation
  out_func=calculate_momos(list(Kresp=k1))
  out_func=out_func[,c("M","RA")]

  # Filter data that contains time points where data is available
  outdf=data.frame(out_func)
  outdf=outdf[outdf$time %in% experimental_data$time,]

  # Evaluate predicted vs experimental residual
  preddf=melt(outdf,id.var="time",variable.name="variables",value.name="values")
  expdf=melt(experimental_data,id.var="time",variable.name="variables",value.name="values")
  ssqres=preddf$values-expdf$values

  return(ssqres)
}
```

Figura 4.15: Creación del vector de residuos

Seguidamente, se establece el parámetro que se desea calibrar, en este caso usamos K_{resp} , después de que el vector de residuos es creado se procede a llamar la función de ajuste, la cual

devuelve la información con el valor ideal para el parámetro indicado.

```
# parameter fitting using levenberg marquart algorithm
# initial guess for parameters
parms=c(Kresp=Kresp)
# fitting
fitval=nls.lm(par=parms,fn=ssq)
print(summary(fitval))
```

Figura 4.16: Encontrar el valor ideal del parámetro Kresp

```
> print(summary(fitval))

Parameters:
      Estimate Std. Error t value Pr(>|t|)
Kresp  0.27879    0.03619   7.704 2.92e-07 ***
```

Figura 4.17: Valor ideal del parámetro Kresp

Finalmente, el valor ideal es enviado como un nuevo parámetro a la función *calculate_momos()* para retornar como un dataframe los nuevos valores de MOMOS calibrados y cercanos a los valores experimentales.

```
> out
      CM      RA
[1,] 50.04000  0.0000
[2,] 95.99952 497.5417
[3,] 127.02085 909.8104
[4,] 158.61259 1240.3441
[5,] 180.25281 1515.0821
[6,] 190.50106 1747.9557
[7,] 188.52944 1949.9086
[8,] 178.35583 2125.1159
[9,] 165.21227 2274.9809
[10,] 152.07010 2401.9059
[11,] 139.95188 2509.2205
[12,] 129.06790 2600.2396
[13,] 119.38400 2677.8530
[14,] 110.80708 2744.4670
[15,] 103.23571 2802.0574
[16,] 96.57274 2852.2396
[17,] 90.72794 2896.3341
[18,] 85.61812 2935.4209
[19,] 81.16674 2970.3841
[20,] 77.30339 3001.9486
[21,] 73.96328 3030.7095
[22,] 71.08690 3057.1562
[23,] 68.61969 3081.6914
[24,] 66.51178 3104.6471
[25,] 64.71781 3126.2977
[26,] 63.19669 3146.8695
[27,] 61.91146 3166.5498
[28,] 60.82908 3185.4934
[29,] 59.92022 3203.8284
[30,] 59.15902 3221.6608
```

Figura 4.18: Dataframe con los datos calibrados de MOMOS

Diseño de las soluciones algorítmicas, para la graficación del modelo momos

El objetivo funcional de *graph_momos()* usando la librería de R *ggplot2* es obtener una representación gráfica de los datos obtenidos durante el desarrollo de la investigación.

```
graph_momos <- function(){
  # Prepare data for graphing
  exp_data=experimental_data
  names(exp_data)=c("time", "CM_experimental", "RA_experimental")

  simulated_data=out_simulated
  names(simulated_data)=c("time", "CM_simulated", "RA_simulated")

  # Overlay calibrated profile with experimental data and simulated data
  tmp_calibrated=melt(out_calibrated,id.var=c("time"),variable.name="variables",value.name="values")
  tmp_experimental=melt(exp_data,id.var=c("time"),variable.name="variables",value.name="values")
  tmp_simulated=melt(simulated_data,id.var=c("time"),variable.name="variables",value.name="values")

  p=ggplot(data=tmp_calibrated,aes(x=time,y=values,color=variables,linetype=variables))+geom_line()
  #p=p+geom_line(data=tmp_experimental,aes(x=time,y=values,color=variables,linetype=variables)) # make lineal to experimental data
  p=p+geom_point(data=tmp_experimental,aes(x=time,y=values,color=variables))
  p=p+geom_line(data=tmp_simulated,aes(x=time,y=values,color=variables,linetype=variables))
  print(p)
}
```

Figura 4.19: Método para graficar los resultados obtenidos

En ella se reflejan las curvas del modelo MOMOS para los diferentes tipos de datos: experimentales, simulados, y calibrados, cabe resaltar que los puntos en la gráfica son la representación de los datos reales que no son sucesivos.

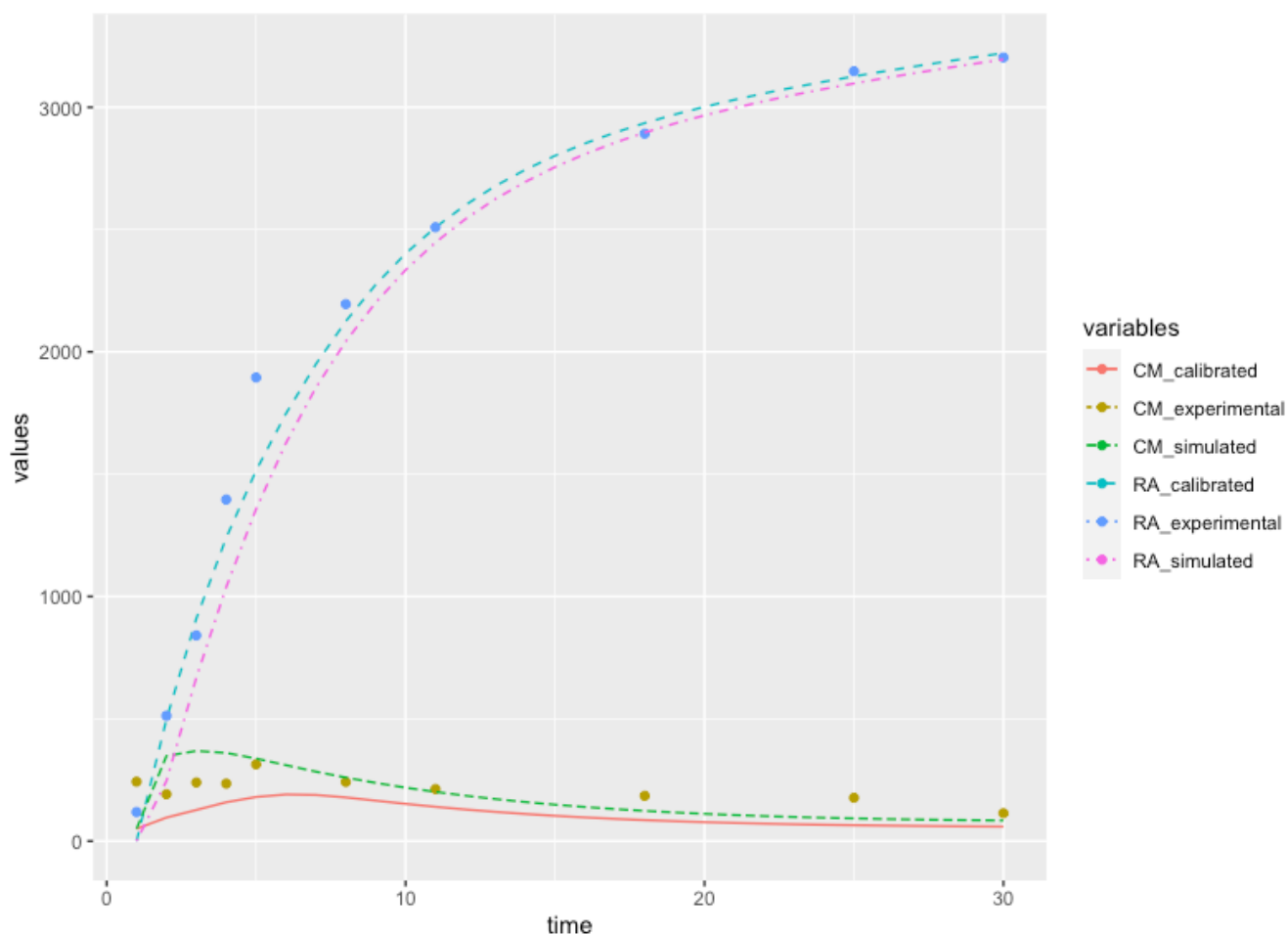


Figura 4.20: Gráfica de los resultados obtenidos

4.4. Pruebas unitarias de las funciones

El proceso de desarrollo de pruebas unitarias consiste en validar las diferentes unidades de funciones desarrolladas, por lo cual, se escribieron usando la librería `testthat` (Wickham, 2017), estas pruebas están contenidas en el archivo `testMomos.R` del directorio `tests/testthat/`, esto permite asegurar que cualquier modificación del código que pueda afectar la salida de alguna función sea detectado en la ejecución de las pruebas.

```

Environment History Connections Build Git Tutorial
Install Test Check More
26 26 63.19669 3146.8695
27 27 61.91146 3166.5498
28 28 60.82908 3185.4934
29 29 59.92022 3203.8284
30 30 59.15902 3221.6608
[1] "===== DATOS DE MOMOS EXPERIMENTALES ====="
  time      CM      RA
1    1 242.7826 118.520
2    2 191.5656 512.245
3    3 239.5755 840.725
4    4 234.9729 1396.190
5    5 313.7010 1895.146
6    8 241.4439 2195.296
7   11 212.7132 2509.596
8   18 184.8797 2892.021
9   25 176.9822 3147.396
10  30 113.9122 3203.661
[1] "===== GRAFICANDO EL MODELO ====="
✓ |      9 | Momos [1.2s]

== Results ==
Duration: 1.2 s

[ FAIL 0 | WARN 0 | SKIP 0 | PASS 9 ]

```

Figura 4.21: Resultado de las pruebas unitarias del paquete

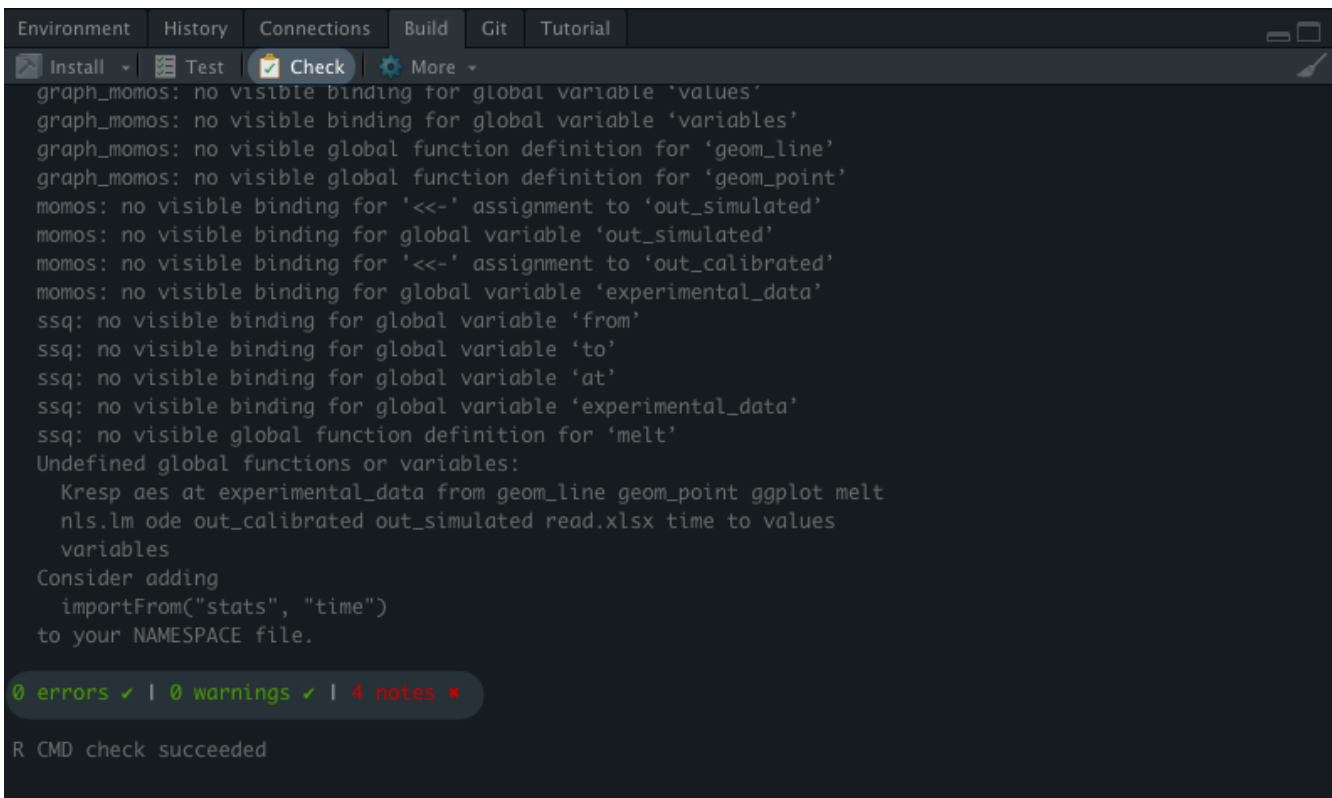
Por otra parte, se valida que las pruebas unitarias creadas cubran la totalidad del código escrito para el paquete, usando el método *test_coverage()* del paquete *devtools* (Wickham, 2017)

File	Lines	Relevant	Covered	Missed	Hits / Line	Coverage
R/momos.R	258	126	126	0	1067	100.00%

Figura 4.22: Cobertura de las pruebas sobre el código creado

4.5. Chequear la carga del paquete

Para realizar el chequeo del paquete se utilizaron las pruebas unitarias proporcionadas por los paquetes RUnit (Zenka, 2015) y testthat (Wickham, 2017), tal como se observa en la Figura 4.22, Para verificar que el código cumple con la estructura de un paquete en R se ejecuta un chequeo desde RStudio, que permite también generar la documentación y creación de las librerías de enlace dinámico (*dynamic link library* DLL), obteniendo como resultado un chequeo sin errores y sin advertencias.

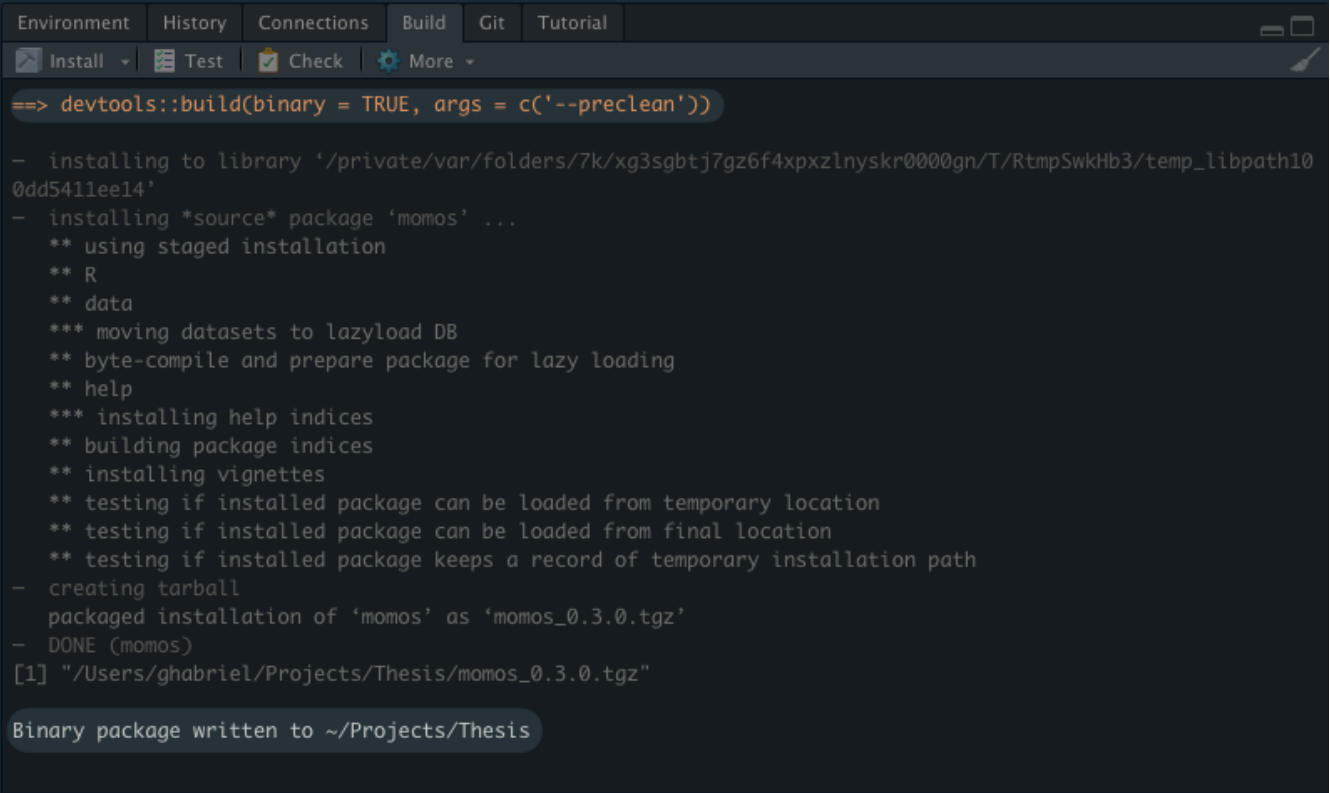


```
Environment History Connections Build Git Tutorial
Install Test Check More
graph_momos: no visible binding for global variable 'values'
graph_momos: no visible binding for global variable 'variables'
graph_momos: no visible global function definition for 'geom_line'
graph_momos: no visible global function definition for 'geom_point'
momos: no visible binding for '<<-' assignment to 'out_simulated'
momos: no visible binding for global variable 'out_simulated'
momos: no visible binding for '<<-' assignment to 'out_calibrated'
momos: no visible binding for global variable 'experimental_data'
ssq: no visible binding for global variable 'from'
ssq: no visible binding for global variable 'to'
ssq: no visible binding for global variable 'at'
ssq: no visible binding for global variable 'experimental_data'
ssq: no visible global function definition for 'melt'
Undefined global functions or variables:
  Kresp aes at experimental_data from geom_line geom_point ggplot melt
  nls.lm ode out_calibrated out_simulated read.xlsx time to values
  variables
Consider adding
  importFrom("stats", "time")
to your NAMESPACE file.
0 errors ✓ | 0 warnings ✓ | 4 notes ✖
R CMD check succeeded
```

Figura 4.23: Resultado del chequeo del paquete

4.6. Construcción del método de distribución del paquete

Para la construcción y distribución del paquete se usan los formatos tar.gz y tgz, que es utilizado por defecto por RStudio, para generar de forma comprimida el código fuente y el binario del paquete respectivamente, tal como se observa en la Figura 4.24 y Figura 4.25.

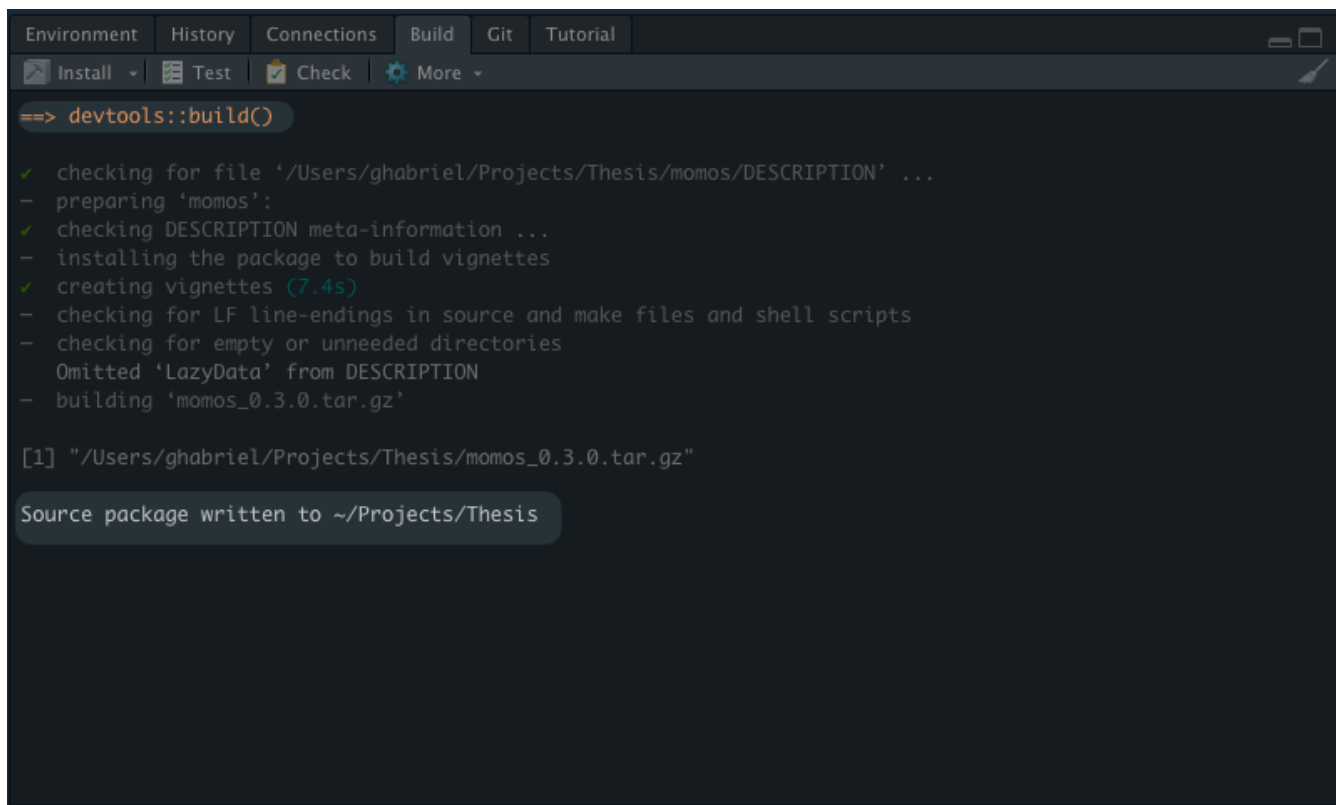


```
Environment History Connections Build Git Tutorial
Install Test Check More
==> devtools::build(binary = TRUE, args = c('--preclean'))

- installing to library '/private/var/folders/7k/xg3sgbtj7gz6f4xpxzlnyskr0000gn/T/RtmpSwkHb3/temp_libpath10
0dd5411ee14'
- installing *source* package 'momos' ...
  ** using staged installation
  ** R
  ** data
  *** moving datasets to lazyload DB
  ** byte-compile and prepare package for lazy loading
  ** help
  *** installing help indices
  ** building package indices
  ** installing vignettes
  ** testing if installed package can be loaded from temporary location
  ** testing if installed package can be loaded from final location
  ** testing if installed package keeps a record of temporary installation path
- creating tarball
  packaged installation of 'momos' as 'momos_0.3.0.tgz'
- DONE (momos)
[1] "/Users/ghabriel/Projects/Thesis/momos_0.3.0.tgz"

Binary package written to ~/Projects/Thesis
```

Figura 4.24: Resultado de la compilación del binario del paquete



```
Environment History Connections Build Git Tutorial
Install Test Check More
==> devtools::build()

✓ checking for file '/Users/ghabriel/Projects/Thesis/momos/DESCRIPTION' ...
- preparing 'momos':
✓ checking DESCRIPTION meta-information ...
- installing the package to build vignettes
✓ creating vignettes (7.4s)
- checking for LF line-endings in source and make files and shell scripts
- checking for empty or unneeded directories
  Omitted 'LazyData' from DESCRIPTION
- building 'momos_0.3.0.tar.gz'

[1] "/Users/ghabriel/Projects/Thesis/momos_0.3.0.tar.gz"

Source package written to ~/Projects/Thesis
```

Figura 4.25: Resultado de la compilación del código fuente del paquete

Finalmente, se utilizó la herramienta *GitHub*, para el control de versiones, así como también la distribución del paquete, disponible en <https://github.com/ghabrielv/momos> dicho repositorio contiene una carpeta llamada *build* donde están alojados los archivos generados Figura 4.26.

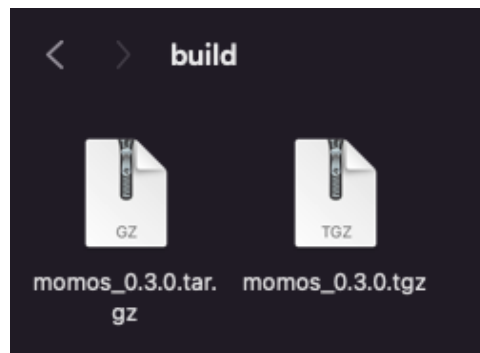


Figura 4.26: Código fuente y binario del paquete

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

Se logró desarrollar el paquete en lenguaje R sobre una herramienta de *Software libre* para simular el modelo MOMOS propuesto por Valery (2018) con la finalidad de sustituir el uso del programa privativo VENSIM.

El estudio de la estructura del modelo MOMOS permitió reescribir las funciones matemáticas como derivadas, ecuaciones diferencias, suma de cuadrados, entre otros en dicho lenguaje.

También fue posible crear pruebas unitarias que cubrieran totalmente el desarrollo planteado, usando diferentes librerías que facilitarán la realización de dichas pruebas, así mismo se llevó a cabo la ejecución de las pruebas funcionales mediante la corrida del paquete con datos simulados y datos reales, dando como resultado unos valores que podían ser ajustados. Para mejorar la diferencia entre los datos se procedió a calibrar el modelo de un parámetro específico *Kresp*.

5.2. Recomendaciones

Se recomienda en una futura investigación calibrar otros parámetros importantes para el modelo.

Adicionalmente, es importante lograr capturar más datos experimentales del modelo para tener un muestreo mayor para mejorar la presión de la simulación.

Referencias Bibliográficas

Martínez Rodríguez, Francisco y Otros: Lombricultura. Manual práctico, Impreso: Unidad de Producciones Gráficas MINREX, La Habana, Cuba, 2003.

Rujano M., Ablan M., Sarmiento L. (2011). Simulación de la respuesta de la materia orgánica del suelo en diferentes ecosistemas ante escenarios de cambio climático en Venezuela. Disponible en: <http://erevistas.saber.ula.ve/index.php/ecodisen/article/view/4372/4149>. Consultado Septiembre 2019.

Universidad Complutense Madrid. Conservación de los recursos naturales para una Agricultura sostenible: Materia orgánica y actividad biológica. Disponible en: <https://www.ucm.es/data/cont/media/www/pag-104576/1.%20Materia%20org%C3%A1nica%20y%20actividad%20biol%C3%B3gica.pdf>. Consultado Septiembre 2019.

Tuomi, M., Vanhala, P., Karhu, K., Fritze, H., and Liski, J. (2008). Heterotrophic soil respiration-comparison of different models describing its temperature dependence. *Ecol.Model.*, 211:182–190.

Valery A.(20xx). La temperatura y humedad como reguladores de la descomposición de la MOS: desempeño de diversas funciones de respuesta en un gradiente altitudinal tropical.

Ferrero R. (2018). Qué es R Software. Disponible en: <https://www.maximaformacion.es/blog-dat/que-es-r-software/>. Consultado Septiembre 2019.

Darghan K. (2018). CGR Paquete en lenguaje R, para el cálculo de índices fisiológicos de crecimiento y componentes del rendimiento en plantas. (Trabajo Especial de Grado de pregrado). Universidad Nacional Experimental del Táchira. San Cristóbal, Estado Táchira.

Contreras R.(2018). DiMBio, paquete en lenguaje R para la discriminación de modelos de simulación dinámicos para procesos biológicos. (Trabajo Especial de Grado de pregrado). Universidad Nacional Experimental del Táchira. San Cristóbal, Estado Táchira.