tinamit Documentation

Release 2.0.2

Julien Malard,

05, 2019

General

1 ¿Quieres contribuir? 3

2 Índice: 5

2.1 Introducción . . . 5
2.2 Instalación . . . 5
2.3 Publicaciones . . . 6
2.4 Agradecimientos . . . 7
2.5 Conectar modelos . . . 8
2.6 Clima . . . 13
2.7 Simulaciones en grupo . . . 14
2.8 Mapas . . . 15
2.9 Uso de datos . . . 18
2.10 Calibraciones . . . 21
2.11 Geografía . . . 23
2.12 Envolturas BF . . . 25
2.13 Envolturas MDS . . . 28
2.14 Traducir . . . 29
2.15 Referencia de envolturas . . . 30
2.16 Interfaz de Programación . . . 30

Python Module Index 53

Index 55

Tinamït es un programa para conectar modelos socioeconómicos (dinámicas de los sistemas) con modelos biofísicos.

Tinamït permite la conexión rápida, flexible y reproducible de modelos con:

- Una estructura fácil para conectar modelos en menos de 10 líneas (en vez de cientos).
- Manejo automático de diferencias en escalas temporales y unidades.
- Conexión con bases de datos para calibración y validación de modelos.
- Inclusión automática de variables de cambios climáticos con la librería (taqdir)
- Una estructura transparente para que todos puedan agregar modelos y recursos.

Note: ¿Encontraste algún error en la documentación? Es culpa mía. Si es error de español, por favor tenga la bondad de corregirlo en GitHub. Si es en otro idioma, corrígelo *así*. ¡Gracias!

CHAPTER 1

¿Quieres contribuir?

Puedes **agregar modelos**, desarrollar **nuevas funcionalidades**, o ayudar a **traducir Tinamït** en tu lengua preferida.

Contacto: Julien Malard: julien.malard@mail.mcgill.ca

CHAPTER 2

Índice:

2.1 Introducción

La modelización DS (Dinámicas de Sistemas) es una manera popular de hacer modelos **participativos** y **visuales** para el manejo ambiental y socio-económico. Vensim y Stella son programas populares y tienen versiones grátis.

No obstante, la inclusión de procesos ambientales o BF (biofísicos) se complica en modelos DS. Y de verdad no deberías tener que contruir un modelo hidrológico o de cultivos en el ambiente DS solamente porque quieres ver los impactos de tu modelo socio-económico DS en el ambiente y vice-versa[1]. Ya existen muchos modelos BF que lo hacen para ti ([2],[3],[4]).

¿Pero cómo conectar un modelo DS con un BF? Por eso existe Tinamït. Empieza con nuestros tutoriales *aquí* o, aún mejor, con la *instalción*.

2.2 Instalación

La instalación debería ser sencilla. Primero necesitarás la versión más recién de Python 3 Después, puedes instalar Tinamït en la terminal con:

```
pip install tinamit
```

Note: Windows todavía está causando complicaciones con programas no escritos en inglés (sí, en 2019). Si encuentras problemas, no hesites en pedir ayuda; por eso existe la comunidad.

Si quieres la versión más recién (en desarrollo), puedes obtenerla de GitHub directamente con:

[1] Jeong H, Adamowski J. 2016. A system dynamics based socio-hydrological model for agricultural wastewater reuse at the watershed scaleAgricultural Water Management, 171: 89-107. https://doi.org/10.1016/j.agwat.2016.03.019

[2] SWAT+. https://swat.tamu.edu/software/plus/

[3] SAHYSMOD. https://www.waterlog.info/sahysmod.htm

[4] Jones, J.W., G. Hoogenboom, C.H. Porter, K.J. Boote, W.D. Batchelor, L.A. Hunt, P.W. Wilkens, U. Singh, A.J. Gijsman, and J.T. Ritchie. 2003. DSSAT Cropping System Model. European Journal of Agronomy 18:235-265.

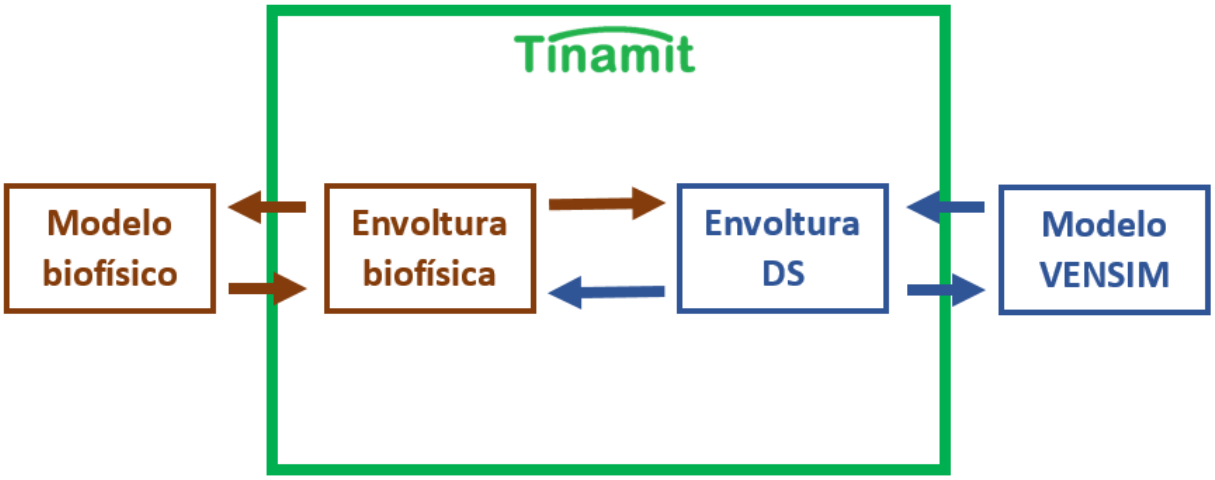


Fig. 1: Figura 1
Diagrama de la estructura conceptual de Tinamït.

```
pip install git+git://github.com/julienmalard/tinamit.git@master
```

Note: Si tienes Windows, es posible que tengas que instalar el `C++ redistributable` de aquí. Toma la versión terminando en `...x86.exe` si tienes Python de 32 bits y en `...x64.exe` si tienes **Python** (no Windows) de 64 bits. Después, instálalo. Por razones obscuras, `SciPy`, un paquete requerido por Tinamït, no funciona en Windows sin éste.

2.3 Publicaciones

Aquí compartimos publicaciones, en todos y cualquier idioma, que decriben o utilizaron Tinamït.

2.3.1 Publicaciones acerca de Tinamït

Publicaciones describiendo Tinamït y sus funcionalidades.

Note: Si publicas algún trabajo el el cual utilizasre Tinamït, puedes citar este artículo. ¡Gracias! :)

- Malard JJ, Inam A, Hassanzadeh E, Adamowski J, Tuy HA, Melgar-Quiñonez H. Development of a software tool for rapid, reproducible, and stakeholder-friendly dynamic coupling of system dynamics and physically-based models. *Environmental Modelling & Software*, 96:410-420. https://doi.org/10.1016/j.envsoft.2017.06.053

2.3.2 Publicaciones con Tinamït

Publicaciones de estudios que utilizaron Tinamït.

- Todavía estamos esperando. ¿No querrías escribir una, por casualidad?

2.4 Agradecimientos

Tinamït es un proyecto de fuente abierta. Las personas e instituciones siguientes hicieron posible su desarrollo.

2.4.1 Autores del código

- Julien Malard
- (Muhammad Azhar Inam Baig)

2.4.2 Autores de envolturas

Gracias a los siguientes por escribir envolturas específicas a los modelos biofísicos siguientes (nota: escribimos las *envolturas* para que estos modelos sean compatibles con Tinamït, ¡no escribimos los modelos sí mismos!)

SAHYSMOD

- (Muhammad Azhar Inam Baig)
- Julien Malard

2.4.3 Traductores

- : (Muhammad Azhar Inam Baig)
- : Julien Malard
- : Julien Malard
- français: Julien Malard

2.4.4 Financiamiento

Los organismos siguientes contribuyeron apoyo financiero a uno o más de los autores del código mientras trabajaban en el desarrollo de Tinamït:

- Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG)
- Centre de recherches pour le développement international du Canada (CRDI)
- Fonds de recherche Nature et technologies du Québec (Canada) (FQRNT)

2.4.5 Instituciones afiliadas

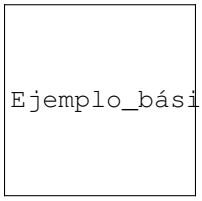
Los autores de Tinamït están afiliados con las instituciones siguientes:

2.5 Conectar modelos

2.5.1 Ejemplo muy básico

Empezaremos con un modelo bastante sencillo. Pero demuestra muy bien cómo funciona Tinamït, y no tienes que instalar cualquier modelo biofísico externo para que te funcione, así que empecemos con este.

Primero vamos a utilizar este modelo de dinámicas de sistemas:

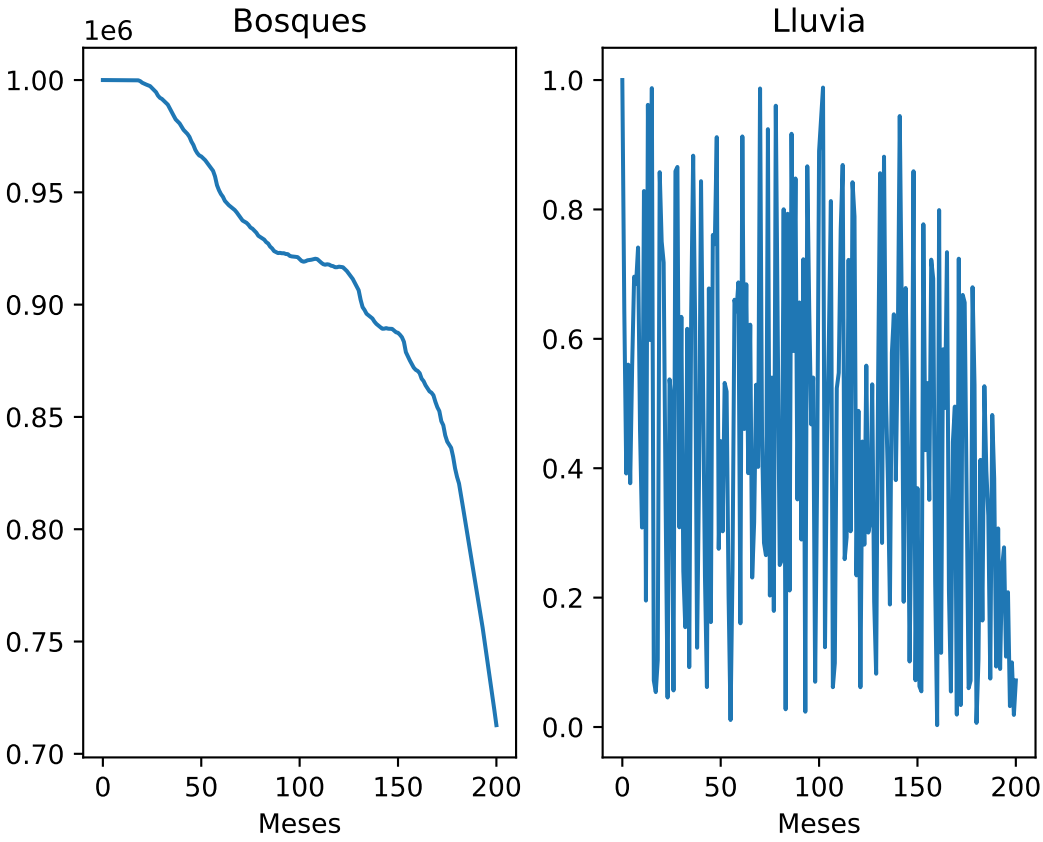


El modelo DS determina, dado la lluvia, la cantidad de pesca posible y su impacto en la necesidad de explotar recursos del bosque.

Del otro lado, el "modelo" biofísico nos da la precipitación según la cubertura forestal.

```
import matplotlib.pyplot as plt

from tinamit.conect import Conectado
from tinamit.ejemplos import obt_ejemplo

mds = obt_ejemplo('sencillo/mds_bosques.mdl')
bf = obt_ejemplo('sencillo/bf_bosques.py')

modelo = Conectado(bf=bf, mds=mds)

# Vamos a conectar los variables necesarios
modelo.conectar(var_mds='Lluvia', var_bf='Lluvia', mds_fuente=False)
modelo.conectar(var_mds='Bosques', var_bf='Bosques', mds_fuente=True)

# Y simulamos
res_conex = modelo.simular(200)

# Visualizar
f, (eje1, eje2) = plt.subplots(1, 2)
eje1.plot(res_conex['mds']['Bosques'].vals)
eje1.set_title('Bosques')
eje1.set_xlabel('Meses')

eje2.plot(res_conex['mds']['Lluvia'].vals)
eje2.set_title('Lluvia')
eje2.set_xlabel('Meses')

eje1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))
```

También comparemos a una corrida sin conexión para ver el impacto de incluir las relaciones entre ambiente y humano.


Bosques
1e6
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
Meses
Lluvia
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
Meses

Note: Siendo subclases de `Modelo`, modelos BF (*ModeloBF*) y DS (*ModeloDS*) también se pueden simular de manera independiente.

```
from tinamit.envolt.mds import gen_mds
from tinamit.envolt.bf import gen_bf

res_mds = gen_mds(mds).simular(200, nombre='Corrida_MDS')
res_bf = gen_bf(bf).simular(200, nombre='Corrida_BF')

# Visualizar
f, (eje1, eje2) = plt.subplots(1, 2)
eje1.plot(res_conex['mds']['Bosques'].vals, label='Conectado')
eje1.plot(res_mds['Bosques'].vals, label='Individual')
eje1.set_title('Bosques')
eje1.set_xlabel('Meses')

eje1.ticklabel_format(axis='y', style='sci', scilimits=(0,0))

eje2.plot(res_conex['mds']['Lluvia'].vals)
eje2.plot(res_bf['Lluvia'].vals)
eje2.set_title('Lluvia')
eje2.set_xlabel('Meses')

f.legend()
```

2.5.2 Opciones de tiempo

Si quieres más control sobre los detalles del eje de tiempo, puedes pasar un objeto `EspecTiempo` a la función `simular()`. Allí puedes especificar no solo el número de paso sino también una fecha inicial (útil para corridas con datos o clima externo), el tamaño de cada paso, y la frequencia con cual se guardan los resultados.

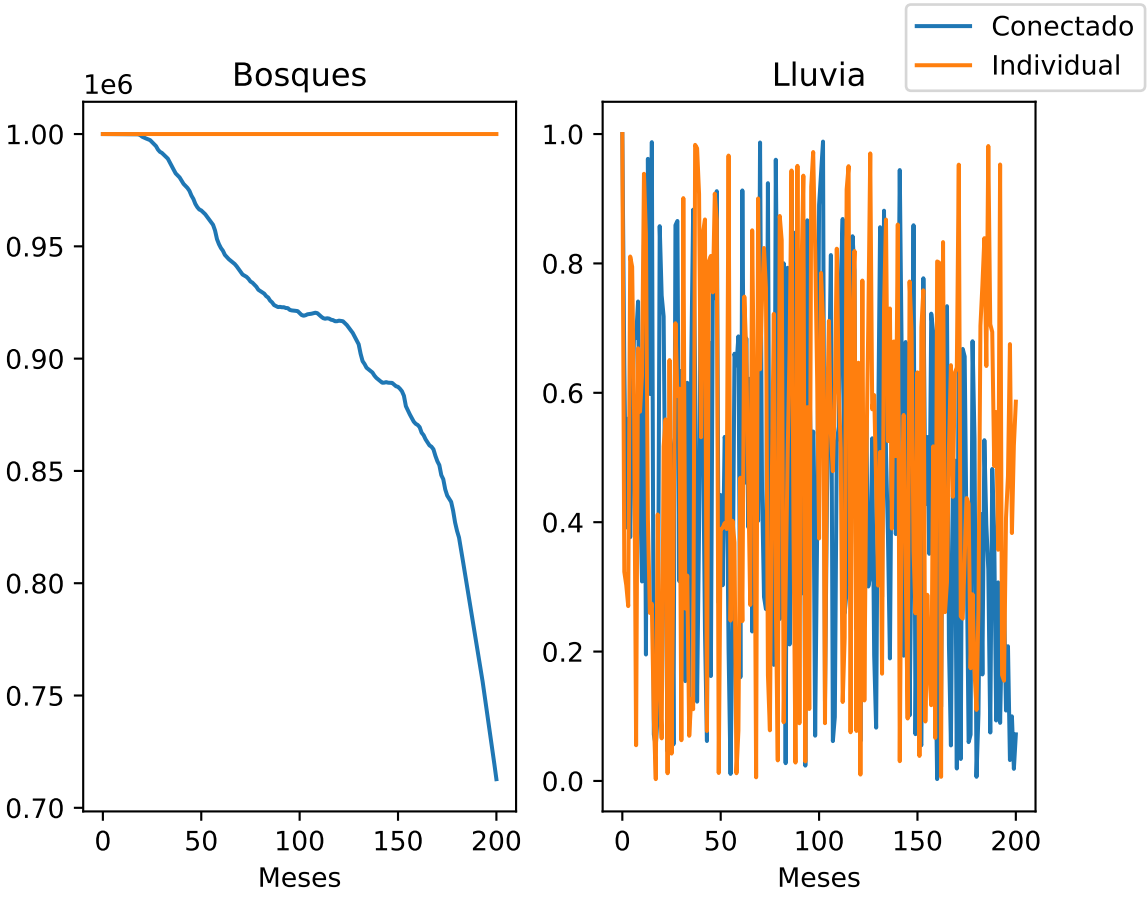
```
from tinamit.tiempo.tiempo import EspecTiempo

t = EspecTiempo(100, f_inic='2000-01-01', tmñ_paso=1, guardar_cada=1)
modelo.simular(t)
```

2.5.3 Unidades de tiempo

Tinamït se encargará de convertir entre unidades de tiempo para ti si difieren entre tus modelos. No obstante, si uno de tus modelos tiene unidad de tiempo no convencional o está en un idioma que Tinamït no reconoce, puede ser que tengas que especificar la conversión manualmente con `nueva_unidad()`, `agregar_trad()` o `agregar_sinónimos()`.

```
from tinamit.unids import nueva_unidad, , agregar_sinónimos

# Una estación tiene 4 meses
nueva_unidad(unid='Estación', ref='Mes', conv=4)

# "día" se dice "" en Tamil
agregar_trad('día', '', leng_trad='', leng_orig='es', guardar=True)
```

(continues on next page)


Conectado
Individual
Bosques
1e6
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0
50
100
150
200
Meses
Lluvia
1.0
0.8
0.6
0.4
0.2
0.0
0
50
100
150
200
Meses

(continued from previous page)

```
# "" también quiere decir "día" en Tamil
agregar_sinónimos('', "", leng='', guardar=True)
```

Tinamït reconoce las unidades de tiempo siguientes: `año`, `mes`, `semana`, `día`, `hora`, `minuto`, `secundo`, `microsecundo`, `millisecundo`, y `nanosecundo`.

2.5.4 3+ modelos

Si tienes más que 2 modelos para conectar, también es fácil con la clase *SuperConectado*. Se pueden conectar de manera horizontal o jerárquica, cómo prefieres.

Horizontal

Se pueden conectar modelos individuales de manera "horizontal" en un solo modelo *SuperConectado*.

```
from tinamit.conectado import SuperConectado

# Crear los 3 modelos
mod_1 = MiModelo1(nombre='modelo 1')
mod_2 = MiModelo2(nombre='modelo 2')
mod_3 = MiModelo3(nombre='modelo 3')

# El Conectado
conectado = SuperConectado([mod_1, mod_2, mod_3])

# Conectar variables entre dos de los modelos por el intermediario del tercero.
conectado.conectar_vars(
    var_fuente='Var 1', modelo_fuente='modelo 1', var_recip='Var 2', modelo_recip=
↪'modelo 2'
)
conectado.conectar_vars(
    var_fuente='Var 2', modelo_fuente='modelo 2', var_recip='Var 3', modelo_recip=
↪'modelo 3'
)

# Simular
res = conectado.simular(10, vars_interés=[mod_1.variables['Var 1'], mod_3.variables[
↪'Var 3']])
```

Los variables `Var 1` del `modelo 1` y `Var 3` del `modelo 3` ahora tendrán valores idénticos a través de la simulación.

Jerárquica

También se pueden anidar modelos adentro de otros.

```
# Los tres modelos
mod_1 = MiModelo1(nombre='modelo 1')
mod_2 = MiModelo2(nombre='modelo 2')
mod_3 = MiModelo3(nombre='modelo 3')

# El primer Conectado
conectado_sub = SuperConectado(nombre='sub', modelos=[mod_1, mod_2])
```

(continues on next page)

(continued from previous page)

```
conectado_sub.conectar_vars(
    var_fuente='Var 1', modelo_fuente='modelo 1', var_recip='Var 2', modelo_recip=
↪'modelo 2'
)

# El segundo Conectado
conectado = SuperConectado([conectado_sub, mod_3])
conectado.conectar_vars(
    var_fuente=mod_2.variables['Var2'], var_recip='Var 3', modelo_recip='modelo 3'
)

# Correr la simulación
res = conectado.simular(10, vars_interés=[mod_1.variables['Var 1'], mod_3.variables[
↪'Var 2']])
```

Este código dará resultados idénticos a los del ejemplo horizontal arriba.

2.6 Clima

Tinamït puede incorporar datos de clima de manera automática, incluso con escenarios de cambios climáticos.

Note: Tinamït emplea `(taqdir)` para obtener datos de cambios climáticos. Si vas a hacer muchas simulaciones con predicciones futuras, se recomienda que leas su documentación.

2.6.1 Especificar variables

Si tienes un variable climático en un modelo DS, puedes especificarlo con la función `conectar_var_clima()`.

```
from tinamit.envolt.mds import gen_mds

mod = gen_mds('Mi modelo.xmile')
mod.conectar_var_clima(var='Lluvia', var_clima='', combin='total', conv=0.001)
mod.conectar_var_clima(var='Temperatura', var_clima='__', combin='prom', conv=1)
```

El parámetro `combin` especifica cómo se deben combinar los datos climáticos de varios días si el modelo se simula con un paso de más de un día. Si es `prom`, se tomará el promedio; si es `total`, se tomará el total de los días incluidos.

Warning: El parámetro `var_clima` **debe** ser un nombre de variable reconocido por taqdir (ver su documentación). Igualmente, si la unidad del variable en tu modelo no corresponde a la unidad del variable en taqdir, tendrás que especificar el factor de conversión en `conv`.

Para modelos BF, la conexión ya debería haberse efectuada en la envoltura específica al modelo, así que no deberías tener que hacer nada.

2.6.2 Correr

Después crearemos un objeto `Clima` para especificar el clima para nuestro lugar. El `escenario` de cambios climáticos sirve para simulaciones del futuro (taqdir obtendrá automáticamente los datos de cambios climáticos; ver aquí).

```
from tinamit.mod.clima import Clima

mi_clima = Clima(lat=31.569, long=74.355, elev=10, escenario='8.5')
t = EspecTiempo(365*50, f_inic='2020-01-01')
mod.simular(t, clima=mi_clima)
```

Si tienes tus propios datos observados, también los puedes incluir en el parámetro `fuentes` que corresponde directamente al parámetro de taqdir.

```
from . import  as Json

fuente = Json('DatosDeMiEstaciónClimáticaPrivadaQueNoVoyACompartirConNadie.json', 31.
↪569, 74.355, 100)

mod.simular(t, clima=Clima(lat=31.569, long=74.355, elev=10, escenario='8.5',␣
↪fuentes=(fuente,)))
```

Te recomendamos que leas la documentación de taqdir si quieres poder aprovechar te todas sus funcionalidades (extensión de datos, interpolación geográfica, desagregación temporal y mucho más).

2.7 Simulaciones en grupo

Si tienes muchas simulaciones para efectuar, puedes ahorar tiempo por hacerlas por grupos con la función `simular_grupo()` y un objeto de simulaciones por grupos (`OpsSimulGrupo`). Igualmente se pueden paralelizar las corridas para ahorar más tiempo.

```
from tinamit.mod import OpsSimulGrupo
from tinamit.envolt.mds import gen_mds

mod = gen_mds('Mi modelo.xmile')

vals_extern = [{'Política 1': 0, 'Política 2': 1}, {'Política 1': 1, 'Política 2': 0}]

ops = OpsSimulGrupo(t=[100, 150], extern=vals_extern)
res = mod.simular_grupo(ops)
```

En el ejemplo arriba, simularemos el modelo con `Política 2` para 100 pasos, y con `Política 1` por 150 pasos.

Warning: Cada opción con valores múltiples debe ser una lista, y cada lista presente en las opciones debe tener el mismo tamaño.

Opciones que no se especificaron en formato de lista se aplicarán a todas las corridas. En el ejemplo abajo, cada política se correrá por 100 pasos.

```
res = mod.simular_grupo(OpsSimulGrupo(t=100, extern=vals_extern))
```

2.7.1 Combinaciones

También se puede ejecutar todas las combinaciones posibles para las opciones de simulación con un objeto `OpsSimulGrupoCombin`. Por ejemplo, puedes simular todas las combinaciones de distintas políticas con varios escenarios de cambios climáticos.

```
from tinamit.mod.clima import Clima

clima_malo = Clima(lat=31.569, long=74.355, elev=10, escenario='2.6')
clima_peor = Clima(lat=31.569, long=74.355, elev=10, escenario='4.5')
clima_fritos = Clima(lat=31.569, long=74.355, elev=10, escenario='8.5')

t = EspecTiempo(365*50, f_inic='2020-01-01')

ops = OpsSimulGrupoCombin(t=t, extern=vals_extern, clima=[clima_malo, clima_peor,␣
↪clima_fritos])
res = mod.simular_grupo(ops)

# Para ver cuáles combinaciones corresponden con cada resultado (en orden)
list(ops)
```

2.8 Mapas

Tinamït viene con algunas funcionalidades para dibujar mapas de resultados de simulación. Todos los mapas están compuestos de objetos *Forma*. Cada *Forma* está vinculada con un archivo `.shp`.

2.8.1 Formas dinámicas

Formas dinámicas (*FormaDinámica*) son las formas cuyos colores varían según los resultados de una simulación. Incluyen *FormaDinámicaNumérica*, la cual toma sus valores en formato de `np.ndarray` o de lista, y *FormaDinámicaNombrada*, la cual quiere sus datos en formato de diccionario.

Los mapas se pueden dibujar desde una matriz de un variable multidimensional en un modelo, o sino de una simulación de grupo donde cada simulación individual representa otra región en el mapa. Ambas situaciones se manejan por *dibujar_mapa_de_res()*.

Simulaciones por grupo

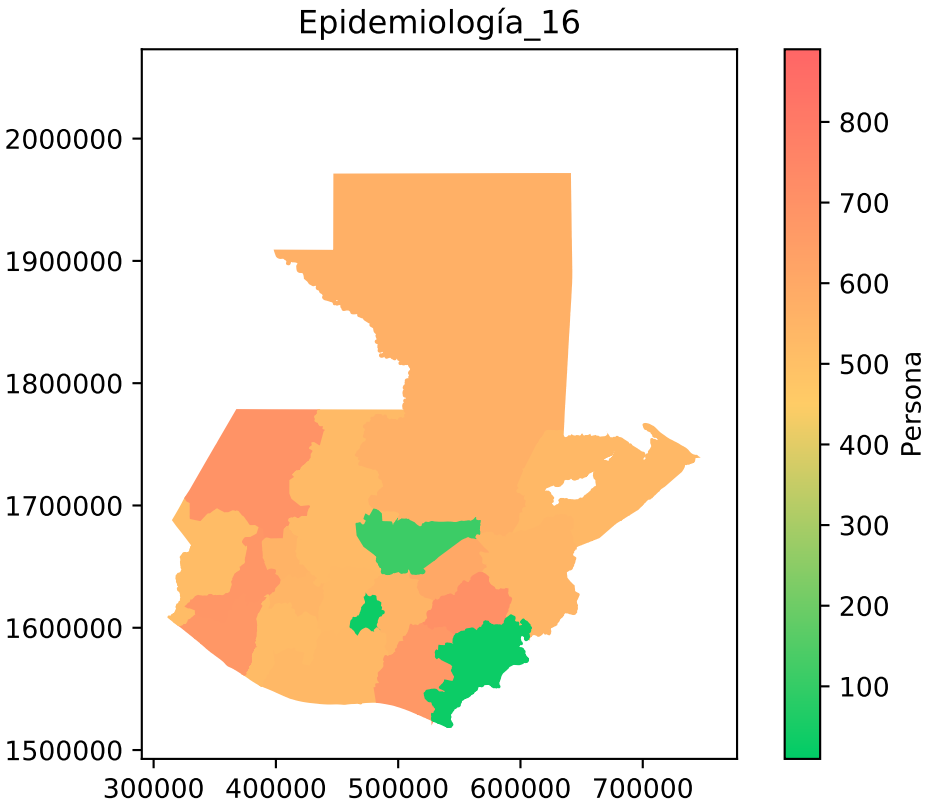
En este ejemplo, correremos un modelo de epidemiología con distintas tazas de contacto para cada departamento de Guatemala.

```
import numpy as np

from tinamit.ejemplos import obt_ejemplo
from tinamit.envolt.mds import gen_mds
from tinamit.geog.mapa import FormaDinámicaNombrada, dibujar_mapa_de_res
from tinamit.mod import OpsSimulGrupo

mds = gen_mds(obt_ejemplo('enfermedad/mod_enferm.mdl'))
forma_deptos = obt_ejemplo('geog_guate/deptos.shp')

ops = OpsSimulGrupo(
    t=50,
    extern=[{'taza de contacto': np.random.random() * 500} for i in range(1, 23)],
    nombre=[str(i) for i in range(1, 23)]
)
res = mds.simular_grupo(ops, nombre='Epidemiología')
```

(continues on next page)

(continued from previous page)

```
frm = FormaDinámicaNombrada(forma_deptos, col_id='COD_DEP', escala_colores=-1)
dibujar_mapa_de_res(forma_dinámica=frm, res=res, var='Individuos Infectados', t=16)
```



Note: El nombre de cada simulación en el grupo debe corresponder con el nombre de una forma en el archivo `.shp` tal como especificado en la columna `col_id`.

Alternativamente, puedes utilizar una *`FormaDinámicaNumérica`*; en ese caso se asiñarán los resultados a las formas según su orden en `OpsSimulGrupo`, nada más.

Variables multidimensionales

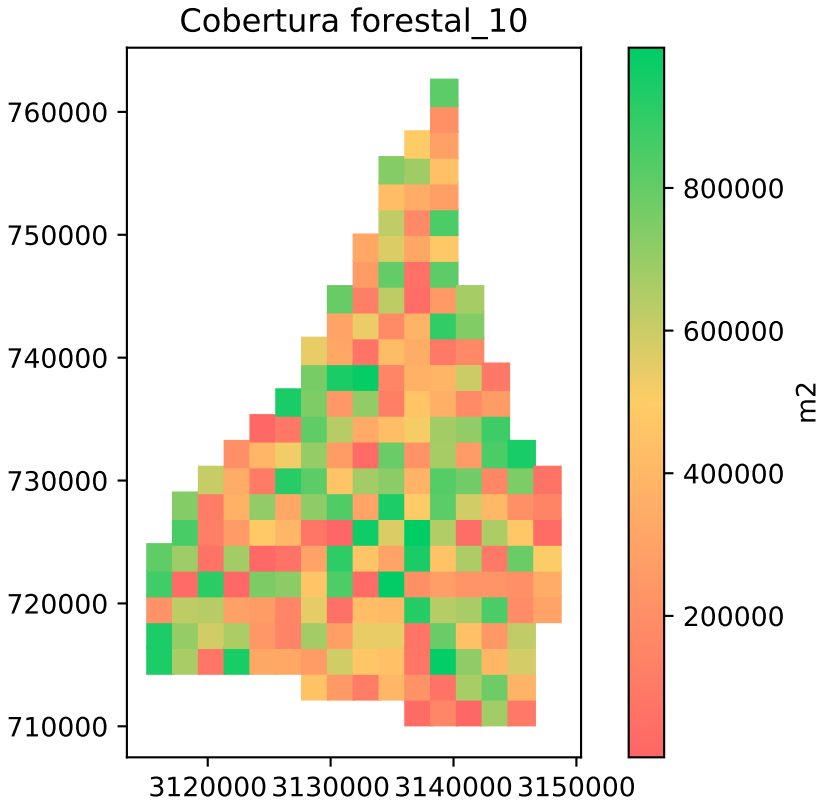
Aplicaremos un modelo sencillo de bosques y lluvia a un mapa de la región del Rechna Doab ( ) en Pakistán. Este mapa divide la región en 215 polígonos, cada cual corresponde a un punto en el variable `Bosque` multidimensional.

```
from tinamit.ejemplos.sencillo.bf_bosques import PruebaBF
from tinamit.geog.mapa import FormaDinámicaNumérica

mod = PruebaBF(215)
polígonos = obt_ejemplo('rechna_doab/polígonos.shp')
```

(continues on next page)

(continued from previous page)

```
extern = {'Bosques': np.random.random(215)*1e6}
res = mod.simular(t=10, extern=extern, nombre='Cobertura forestal')

frm = FormaDinámicaNumérica(polígonos, col_id='Id')
dibujar_mapa_de_res(forma_dinámica=frm, res=res, var='Bosques', t=10)
```



2.8.2 Formas estáticas

También puedes agregar formas estáticas (*FormaEstática*), que no dependen de los resultados de una simulación y que se agregan solamente por razones estéticas.

Por el momento, tienes:

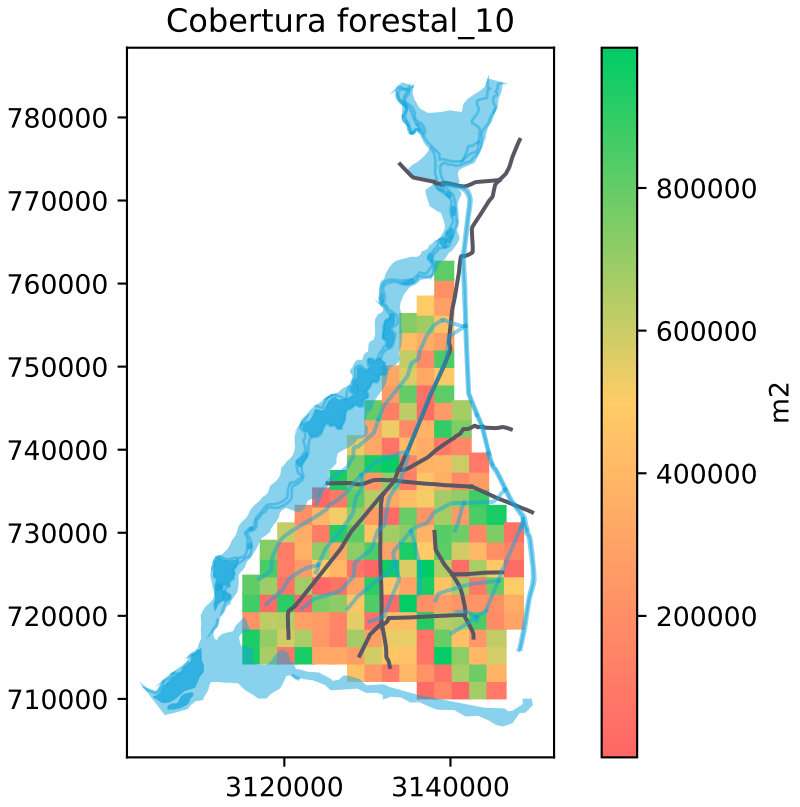
- Cuerpos de agua: *Agua*
- Bosques: *Bosque*
- Calles: *Calle*
- Zonas urbanas: *Ciudad*

```
from tinamit.geog.mapa import Agua, Calle

calles = Calle(obt_ejemplo('rechna_doab/calle.shp'))
```

(continues on next page)

(continued from previous page)

```
río = Agua(obt_ejemplo('rechna_doab/río.shp'))
canales = Agua(obt_ejemplo('rechna_doab/canal.shp'), llenar=False)
dibujar_mapa_de_res(forma_dinámica=frm, otras_formas=[calles, canales, río], res=res,␣
↪var='Bosques', t=10)
```



2.9 Uso de datos

Puedes utilizar datos externos en Tinamït para especificar valores de variables en simulaciones, para alimentar calibraciones, y para efectuar validaciones.

2.9.1 Datos exógenos

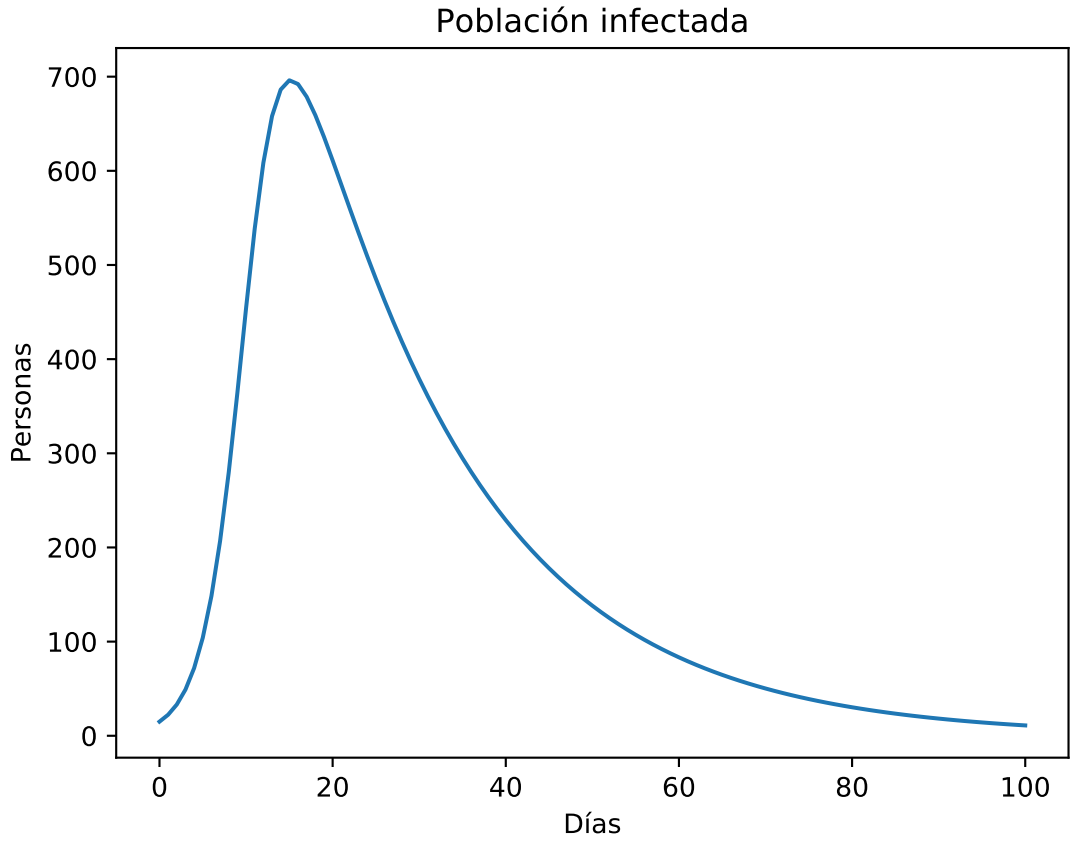
Puedes especificar valores de parámetros o de variables externos en el transcurso de una simulación. Aquí vamos a hacer una simulación con un modelo sencillo de contagión de una enfermedad.

```
import matplotlib.pyplot as plt

from tinamit.ejemplos import obt_ejemplo
from tinamit.envolt.mds import gen_mds
```

(continues on next page)

(continued from previous page)

```
mds = gen_mds(obt_ejemplo('enfermedad/mod_enferm.mdl'))

extern = {'número inicial infectado': 15}
res = mds.simular(t=100, extern=extern)

# Visualizar
plt.plot(res['Individuos Infectados'].vals)
plt.title('Población infectada')
plt.xlabel('Días')
plt.ylabel('Personas')
```



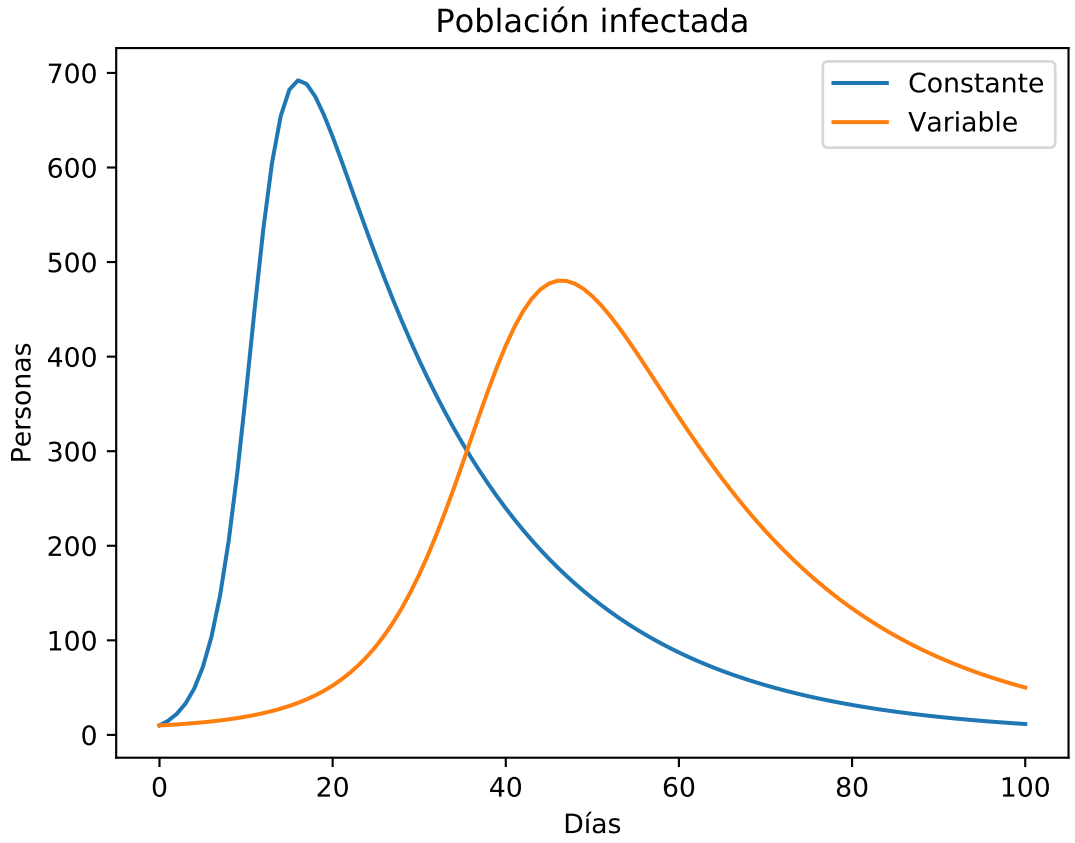
Igualmente podemos pasar datos que varían temporalmente. Por ejemplo, la taza de infección puede variar a través de la epidemía.

```
from tinamit.tiempo import EspecTiempo
import numpy as np
import pandas as pd

extern = pd.DataFrame(
    data={'taza de infección': np.arange(0.001, 0.005, (0.005-0.001)/100)},
    index=pd.date_range('2000-01-01', periods=100)
)
res_base = mds.simular(t=EspecTiempo(100, f_inic='2000-01-01'))
```

(continues on next page)

(continued from previous page)

```
res_extern = mds.simular(t=EspecTiempo(100, f_inic='2000-01-01'), extern=extern)

# Visualizar
plt.plot(res_base['Individuos Infectados'].vals, label='Constante')
plt.plot(res_extern['Individuos Infectados'].vals, label='Variable')

plt.legend()
plt.title('Población infectada')
plt.xlabel('Días')
plt.ylabel('Personas')
```



Si quieres más control sobre el uso de variables externos, puedes utilizar la función `gen_extern()` para generar un objeto `Extern` que puedes pasar al parámetro `extern`.

Note: Se puede utilizar datos en formato de `dict`, `pd.DataFrame` o `xr.Dataset`.

2.9.2 Bases de datos

Para calibraciones y validaciones, puedes utilizar bases de datos, las cuales te permiten combinar varias fuentes de datos además de especificar datos geográficos.

En Tinamït, una base de datos (*BD*) está compuesta de una o más fuentes (*Fuente*). Fuentes pueden representar datos en formato `.csv` (*FuenteCSV*), diccionarios (*FuenteDic*), `Dataset` o `DataArray` de `xarray` (*FuenteBaseXarray* y *FuenteVarXarray*, respectivamente), o un `DataFrame` de `pandas` (*FuentePandas*).

Ver *Calibraciones* y *Geografía* para más detalles.

2.10 Calibraciones

2.10.1 Calibrar modelos

Tinamït puede calibrar modelos según variables observados. Las calibraciones se efectuan con calibradores (`CalibradorMod`), por ejemplo, `CalibradorModSpotPy`.

```
import numpy as np

from tinamit.ejemplos import obt_ejemplo
from tinamit.envolt.mds import gen_mds

mod = gen_mds(obt_ejemplo('enfermedad/mod_enferm.mdl'))
```

Generaremos unos datos artificiales (sí, hacemos trampa).

```
from tinamit.datos.fuente import FuenteDic

paráms = {
    'taza de contacto': 81.25,
    'taza de infección': 0.007,
    'número inicial infectado': 22.5,
    'taza de recuperación': 0.0375
}

simul = mod.simular(
    t=100, extern=paráms,
    vars_interés=['Individuos Suceptibles', 'Individuos Infectados', 'Individuos␣
↪Resistentes']
)
datos = FuenteDic({ll: v[:, 0] for ll, v in simul.a_dic().items()}, nombre='Datos',␣
↪fechas=np.arange(101))
```

Y efectuamos la calibración.

```
from tinamit.calibs.mod import CalibradorModSpotPy

líms_paráms={
    'taza de contacto': (0, 100),
    'taza de infección': (0, 0.02),
    'número inicial infectado': (0, 50),
    'taza de recuperación': (0, 0.1)
}

calibs = CalibradorModSpotPy(mod).calibrar(líms_paráms=líms_paráms, datos=datos, n_
↪iter=50)
```

2.10.2 Calibrar ecuaciones

En el caso de modelos de dinámicas de sistemas, también se pueden calibrar los parámetros de ecuaciones individuales si tienes los datos necesarios.

Las calibraciones se pueden hacer con optimización (*CalibradorEcOpt*) o con inferencia bayesiana (*CalibradorEcBayes*).

Note: Casos sencillos con muchos datos disponibles generalmente se pueden resolver mucho más rápido con optimización normal que con la más sofisticada inferencia bayesiana.

El el modelo epidemiológico, el número de contactos con susceptibles se determina por el número de suceptibles y la taza de contacto según la ecuación `contactos con suceptibles = Individuos Suceptibles * taza de contacto`. Suponiendo que tenemos datos para el número de suceptibles y el número de contactos, podemos estimar la taza de contacto.

```
from tinamit.calibs.ec import CalibradorEcOpt
from tinamit.datos.bd import BD
from tinamit.datos.fuente import FuenteDic

n_obs = 100
taza_contacto = 125
individuos_suceptibles = np.random.random(n_obs)

contactos_con_suceptibles = individuos_suceptibles * taza_contacto + np.random.
↪normal(0, 1, n_obs)
bd = BD(
    fuentes=FuenteDic({
            'contactos con suceptibles': contactos_con_suceptibles,
            'Individuos Suceptibles': individuos_suceptibles,
            'f': np.arange(n_obs)
        },
        nombre='Datos generados',
        fechas='f'
    )
)

calibrador = CalibradorEcOpt(
    ec=mod.variables['contactos con suceptibles'].ec, nombre='contactos con␣
↪suceptibles',
    paráms=['taza de contacto']
)
calib_ec = calibrador.calibrar(líms_paráms={'taza de contacto': (0, 200)}, bd=bd)
```

2.10.3 Validar

Por supuesto, no hay calibración sin validación. (Al menos que tengas que publicar ya.) Las validaciones se efectuan con *ValidadorMod*.

```
from tinamit.calibs.valid import ValidadorMod

valid = ValidadorMod(mod).validar(
    t=100, datos=datos, paráms={prm: trz['mejor'] for prm, trz in calibs.items()}
)
```

2.11 Geografía

Tinamït cuenta con funcionalidades de datos geográficos para simulaciones, calibraciones y validaciones.

2.11.1 Especificación

Primero tenemos que especificar nuestra geografía. Ésta está compuesta de lugares (*Lugar*) de distintos niveles (*Nivel*). Por ejemplo, en el nivel departamental podremos encontrar varios muninicipios.

```
from tinamit.geog.región import Nivel, Lugar

muni = Nivel('Municipio')
dept = Nivel('Departamento', subniveles=muni)
terr = Nivel('Territorio', subniveles=muni)
país = Nivel('País', subniveles=[dept, terr])

muni1, muni2, muni3 = [Lugar('Muni%i' % i, nivel=muni, cód='M' + str(i)) for i in␣
↪range(1, 4)]

dept1 = Lugar('Dept1', nivel=dept, cód='D1', sub_lugares=[muni1, muni2])
dept2 = Lugar('Dept2', nivel=dept, cód='D2', sub_lugares=[muni3])
terr1 = Lugar('Terr1', nivel=terr, cód='T1', sub_lugares=[muni1])
terr2 = Lugar('Terr2', nivel=terr, cód='T2', sub_lugares=[muni2])

guate = Lugar(
    'Guatemala', sub_lugares={muni1, muni2, muni3, dept1, dept2, terr1, terr2},
    nivel=país
)
```

O, para ahorar tiempo con geografías más complejas, puedes emplear la función *gen_lugares()*, que genera un lugar automáticamente a base de un archivo de `.csv`.

```
from tinamit.geog.región import gen_lugares
from tinamit.ejemplos import obt_ejemplo

guate = gen_lugares(obt_ejemplo('geog_guate/geog_guate.csv'), nivel_base='País',␣
↪nombre='Iximulew')
```

Note: Puedes especificar niveles paralelos. Por ejemplo, aquí `Departamento` y `Territorio` son dos maneras alternativas de agrupar los municipios de Guatemala.

2.11.2 Calibración

Se pueden calibrar modelos según datos geográficos. El resultado será una calibración distinta para cada lugar para el cual tienes datos.

Ecuaciones

Tinamït tiene funcionalidades **experimentales** para calibrar ecuaciones con inferencia bayesiana jerárquica. Esta funcionalidad permite al modelo inferir valores el regiones para las cuales tienes muy poco (o hacia no) datos. Funciona

por calibrar los variables al nivel más alto (por ejemplo, nacional) y después ajustar sus estimos para cada sublugar según la disponibilidad de datos.

Cada *Nivel* en tu geografía corresponderá a un nivel distinto en el modelo jerárquico.

> **Warning:** La calibración con inferencia bayesiana jerárquica es muy emocionante pero también todavía **experimental**.
>
> Si tus ecuaciones no están bien definidas o si su forma no corresponde con la de los datos, correrá muy lentamente la calibración y tus resultados no valdrán nada de todo modo. Siempre es buena idea visualmente comparar los resultados con los datos.

Simplemente puedes pasar un objeto *Lugar* a *CalibradorEcOpt* o al *CalibradorEcBayes* (ver *Calibraciones*).

Modelos

Calibraciones geográficas se pueden también aplicar al nivel del modelo entero.

```
import numpy as np

from tinamit.calibs.geog_mod import SimuladorGeog, CalibradorGeog
from tinamit.datos.bd import BD
from tinamit.datos.fuente import FuenteDic

paráms = {
          '708': {
              'taza de contacto': 81.25, 'taza de infección': 0.007, 'número inicial␣
↪infectado': 22.5,
              'taza de recuperación': 0.0375
          },
          '1010': {
              'taza de contacto': 50, 'taza de infección': 0.005, 'número inicial␣
↪infectado': 40,
              'taza de recuperación': 0.050
          }
}

# Unos datos artificiales
simul = SimuladorGeog(mds).simular(
    t=100, vals_geog=paráms,
    vars_interés=['Individuos Suceptibles', 'Individuos Infectados', 'Individuos␣
↪Resistentes']
)
datos = {
    lg: {ll: v[:, 0] for ll, v in simul[lg].a_dic().items()} for lg in paráms
}

datos = BD([
    FuenteDic(datos[lg], 'Datos geográficos', lugares=lg, fechas=np.arange(101)) for␣
↪lg in paráms
])

calib = CalibradorGeog(mds).calibrar(t=100, datos=datos, líms_paráms=líms_paráms, n_
↪iter=50)
```

2.11.3 Validación

Se puede validar una calibración geográfica con la clase *ValidadorGeog*.

```
from tinamit.calibs.geog_mod import ValidadorGeog

valid = ValidadorGeog(mds).validar(
         t=100, datos=datos,
         paráms={lg: {prm: trz['mejor'] for prm, trz in calib[lg].items()} for lg in␣
↪paráms}
)
```

2.12 Envolturas BF

Cada modelo BF requiere la presencia de una envoltura especial que maneja su interacción (simulación, intercambio de variables) con Tinamït.

Note: Todas las envolturas son subclases de *ModeloBF*. No obstante, Tinamït viene con clases especiales para simplificar tu vida con casos de modelos más compicados.

¿Cómo escoger la clase pariente? Si tu modelo da resultados con el mismo paso de tiempo con el cual puede avanzar (por ejemplo, da resultados mensuales y avanza con paso mensual), entonces es un *modelo sencillo*. Si da resultados con paso más pequeño que el paso con el cual puede avanzar (por ejemplo, un modelo hidrológico que simula 1 año a la vez pero después devuelve resultados diarios), entonces es un *modelo determinado*.

Si tu modelo tiene subdivisiones temporales adicionales (p. ej., SAHYSMOD simula por un año, pero después devuelve datos por *estaciones* de duración de entre 1 a 12 meses), entonces es un *modelo bloques*.

Y, por fin, si no sabes antes de simular cuánto tiempo va a simular (p. ej., modelos de cultivos que corren hacia la cosecha), entonces tienes un *modelo indeterminado*.

2.12.1 Modelos Sencillos

Siendo modelos sencillos, las envolturas basadas directamente en *ModeloBF* solamente deben implementar las funciones siguientes:

1. `unidad_tiempo()`: Devuelve la unidad de tiempo del modelo.
2. `incrementar()`: Avanza el modelo.
3. `__init__()`: Inicializa el modelo. En la llamada a `super().__init__` debes incluir un objeto `VariablesMod` con los variables del modelo.

Funciones y atributos opcionales:

1. `paralelizable()`: Indica si el modelo se puede paralelizar para ahorar tiempo.
2. `iniciar_modelo()`: Acciones llamadas justo antes de la simulación.
3. `cerrar()`: Efectua acciones de limpieza al final de una simulación.
4. `_correr_hasta_final()`: Permite el modelo de combinar pasos de simulación cuando posible para ser más rápido.
5. `instalado()`: Verifica si el modelo correspondiendo a la envoltura está instalado en la computadora o no.
6. `Modelo.idioma_orig`: Indica el idioma de los nombres de variables del modelo.

Warning: Tu implementación de `incrementar()` **debe** incluir una llamada a `super().incrementar(rebanada)` al final para que valores de parámetros externos y de clima se actualicen correctamente. Igualmente, cualquier reimplementación de `iniciar_modelo()` **debe** incluir una llamada a `super().iniciar_modelo(corrida)` al final.

En la función `incrementar()`, se puede acceder los variables del modelo con *símismo.variables["nombre del variable"]*, obtener su valor con `obt_val()`, y cambiar su valor con `poner_val()`:

```
lago = símismo.variables['Lago']
val_lago = lago.obt_val()

nuevo_valor = 100
lago.poner_val(nuevo_valor)
```

2.12.2 Modelos Determinados

Modelos determinados (*ModeloDeterminado*) simulan por un periodo fijo, y después devuelven egresos de manera retroactiva. Muchos modelos biofísicos (SWAT, DSSAT, STICS) funcionan (o pueden funcionar) así.

El paso del modelo sigue siendo la unidad de tiempo de los egresos (p. ej., días), y se agrega el concepto de un `ciclo`, o el tiempo mínimo que se puede efectuar una simulación (p. ej., 1 año).

Funciones obligatorias:

1. `unidad_tiempo()`: Devuelve la unidad de tiempo de los **egresos** del modelo.
2. `avanzar_modelo()`: Avanza el modelo de un cierto número de **ciclos**.
3. `__init__()`: Inicializa el modelo. En la llamada a *super().__init__* debes incluir un objeto *VariablesModDeter* con los variables del modelo.

Note: No se implementa `incrementar()` en modelos determinados. Tinamït lo implementa automáticamente y llama `avanzar_modelo()` en los momentos oportunos de la simulación.

Modelos determinados pueden tener variables que cambian con el paso (*VarPasoDeter*) y otros que cambian con el ciclo (`Variable`). Ambos se pueden pasar al *VariablesModDeter* de la inicialización.

Para cambiar los valores de *VarPasoDeter* en la función `avanzar_modelo()`, se llama `poner_vals_paso` con una matriz de valores para todos los pasos en el ciclo presente. Para obtener su valor en el paso actual, se llama `obt_val`, o sino `obt_vals_paso` para obtener la matriz de sus valores para todos los pasos en el ciclo actual.

Note: Tinamït se encarga de actualizar los valores de los variables por paso según el paso actual del modelo.

Igualmente pueden implementar todas las funciones opcionales de *ModeloBF*.

2.12.3 Modelos Bloques

Modelos bloques (*ModeloBloques*) son una subclase de (*ModeloDeterminado*). Además de pasos y ciclos, tienen el concepto de *bloques*. En su simulación, un ciclo contiene varios bloques hechos de cantidades variables de pasos.

Funciones obligatorias:

1. `unidad_tiempo()`: Devuelve la unidad de tiempo de **base** de los **egresos** del modelo. Por ejemplo, si el modelo simula por año y devuelve datos por tres estaciones de 4, 5 y 3 meses, entonces la unidad de tiempo sería *mes*.
2. `avanzar_modelo()`: Avanza el modelo de un cierto número de **ciclos**.
3. `__init__()`: Inicializa el modelo. En la llamada a *super().__init__* debes incluir un objeto *VariablesModBloques* con los variables del modelo.

Modelos bloques pueden tener variables bloques (*VariablesModBloques*), igual que variables que cambian con el paso (*VarPasoDeter*) y otros que cambian con el ciclo (`Variable`).

Note: Tinamït actualiza automáticamente el paso, el bloque y el ciclo de sus variables (con los valores, por supuesto).

Igualmente pueden implementar todas las funciones opcionales de *ModeloBF*.

2.12.4 Modelos Indeterminados

Modelos indeterminados (*ModeloIndeterminado*) avanzan por periodos de tiempo indeterminados cada vez que se simulan. Tienen el concepto de ciclos, pero el tamaño del ciclo varia entre simulaciones.

Funciones obligatorias:

1. `unidad_tiempo()`: Devuelve la unidad de tiempo de los **egresos** del modelo.
2. `mandar_modelo()`: Avanza el modelo.
3. `__init__()`: Inicializa el modelo. En la llamada a *super().__init__* debes incluir un objeto *VariablesModIndeterminado* con los variables del modelo.

En *VariablesModIndeterminado*, se pueden incluir variables cuyos valores cambian con el paso (*VarPasoIndeter*), tanto como variables cuyos valores quedan constantes adentro del mismo ciclo (`Variable`).

En `mandar_modelo()`, se puede utilizar las mismas funciones que con modelos determinados para establecer y acceder los valores de los variables.

Igualmente pueden implementar todas las funciones opcionales de *ModeloBF*.

2.12.5 Variables clima

Si tu modelo incluye variables climáticos, puedes especificarlos con la función `conectar_var_clima()` en el *__init__()* de la clase. Tinamït se encargará de la actualización del valor del variables cuando se efectua una simulación con clima activado.

Note: Si tu modelo requiere datos de manera más sofisticada (por ejemplo, DSSAT debe guardar en un archivo externo todos los datos climáticos *antes* de empezar la simulación), puedes acceder el objeto de `Clima` de la corrida actual (si hay) con *símismo.corrida.clima* y llamar sus funciones `obt_datos()` o `obt_todos_vals()`.

2.12.6 Configuración

Puedes incluir variables de configuración en tu envoltura (p. ej., la ubicación de un archivo ejecutable). Se obtiene el valor con `obt_conf()`, y usuarias pueden establecer su valor con *MiEnvoltura.estab_conf("llave", "valor")*. Por ejemplo:

```
from tinamit.envolt.sahysmod.bf import ModeloSAHYSMOD
ModeloSAHYSMOD.estab_conf("exe", "C:\\Camino\\hacia\\mi\\SAHYSMODConsole.exe")
```

2.12.7 Pruebas

Siempre es buena idea tener pruebas para saber si tu envoltura funciona bien o no. Tinamït te permite integrar pruebas de lectura de datos, de lectura de egresos y de simulación con tus envolturas.

Puedes implementar las funciones *prb_ingreso()*, *prb_egreso()*, o *prb_simul()* para tu modelo.

Después, puedes integrar las funciones *verificar_leer_ingr()*, *verificar_leer_egr()*, y *verificar_simul()* con tus pruebas automáticas para comprobar que todo están bien con tu envoltura. La primera vez que corren las pruebas, Tinamït guardará en el disco los resultados de la lectura de datos y de la simulación. Asegúrate que estén correctos los variables. Si, en el futuro, tu envoltura ya no da los mismos resultados, Tinamït te avisará de un error.

Note: Estas funciones se aplican automáticamente a todas las envolturas incluidas con la distribución de Tinamït.

2.12.8 Distribución

Puedes compartir tu nueva envoltura como paquete Python independiente. Igualmente puedes contribuirlo al código fuente de Tinamït, después de cual todas las usuarias de Tinamït podrán acceder tu envoltura.

2.13 Envolturas MDS

Envolturas para modelos de dinámicas de sistemas son subclases de `ModeloMDS`.

Note: Las envolturas para modelos DS son **universales**. Es decir, la misma envoltura funcionará para todos los modelos creados con el mismo programa (p. ej., Vensim), no importe el contenido del modelo sí mismo.

2.13.1 Cómo crear tu envoltura

Funciones y atributos para implementar:

1. `unidad_tiempo()`: Devuelve la unidad de tiempo del modelo.
2. `incrementar()`: Avanza el modelo.
3. `__init__()`: Inicializa el modelo. En la llamada a `super().__init__` debes incluir un objeto *VariablesMDS* con los variables del modelo.
4. `cambiar_vals()`: No estríctamente necesario, pero la casi totalidad de modelos DS necesitarán tomar acción específica para cambiar valores de variables en el modelo externo.
5. `ModeloDS.ext`: Una lista de las extensiones de archivo que se pueden leer por la envoltura.

Funciones y atributos opcionales:

1. `paralelizable()`: Indica si el modelo se puede paralelizar para ahorar tiempo.
2. `iniciar_modelo()`: Acciones llamadas justo antes de la simulación.

3. `cerrar()`: Efectua acciones de limpieza al final de una simulación.
4. `_correr_hasta_final()`: Permite el modelo de combinar pasos de simulación cuando posible para ser más rápido.
5. `instalado()`: Verifica si el modelo correspondiendo a la envoltura está instalado en la computadora o no.

> **Warning:** Tu implementación de `incrementar()` **debe** incluir una llamada a `super().incrementar(rebanada)` al final para que valores de parámetros externos y de clima se actualicen correctamente. Igualmente, cualquier reimplementación de `iniciar_modelo()` **debe** incluir una llamada a `super().iniciar_modelo(corrida)` al final, y `cambiar_vals()` una a `super().cambiar_vals(valores)`.

Cada variable en *VariablesMDS* debe ser uno de *VarConstante*, *VarInic*, *VarNivel*, o :class:‘~tinamit.envolt.mds.VarAuxiliar.

2.13.2 Autogeneración

La función `gen_mds()` de Tinamït puede escoger automáticamente la envoltura más apropriada para un archivo dado de modelo DS según el atributo *ModeloMDS.ext* de cada clase de envoltura. Puedes llamar la función `registrar_envolt_mds()` para registrar tu nueva clase de modelo DS en Tinamït, y `olvidar_envolt_mds()` para quitarla.

Si estás modificando el código fuente de Tinamït, puedes agregar tu clase a *tinamit.envolt.mds._auto._subclases* para que se tome automáticamente en cuenta.

2.13.3 Distribución

Puedes compartir tu nueva envoltura como paquete Python independiente. Igualmente puedes contribuirlo al código fuente de Tinamït, después de cual todas las usuarias de Tinamït podrán acceder tu envoltura.

2.14 Traducir

¿Quieres leer este en tu propia lengua? La documentación de Tinamït se traduce de manera colaborativa. Puedes utilizar el servidor Transifex o Zanata (syncronizamos los dos, así que no hay problema cuál eliges):

- Zanata: Empezar ahora.
- Transifex: Empezar ahora.

2.14.1 Para manejadoras

El página de integración contínua Travis CI de Tinamït está configurada para automáticamente mandar cambios en el texto para traducir a Transifex, y para obtener las nuevas traducciones disponibles en Transifex, cada vez que se hace un cambio a la rama `master` del proyecto en GitHub.

Las nuevas traducciones se mandarán por Travis CI a una nueva rama llamada `Transifex` en GitHub; puedes incorporarla con la rama `master` y después aparecerán automágicamente las traducciones en la documentación en línea.

2.15 Referencia de envolturas

Abajo puedes encontrar información sobre todas las envolturas que vienen con Tinamït. Si quieres agregar una nueva, contáctenos y la podremos *desarrollar juntos*.

2.15.1 Modelos DS

Envolturas incluidas en Tinamït:

- **PySD**: Implementación de modelos DS en puro Python. Más rápido que Vensim, pero no incluye todas las funcionalidades (por el momento). Ver su documentación aquí. Puede leer modelos en formato `.mdl` de Vensim, tanto como el estándar `.xmile` (utilizado, entre otros, por Stella).
- **Vensim**: Un programa de MDS bastante popular. Desafortunadamente, requiere la versión pagada (DSS) para conectar con Tinamït. Ver su página oficial. Además, solamente funciona en Windows. Con Vensim, primero hay que ir a cada variable en tu modelo Vensim que quieres que pueda recibir valores desde el modelo biofísico y escoger `Gaming` como tipo de variable. Después, hay que publicar el modelo en formato `.vpm`.

2.15.2 Modelos BF

Envolturas incluidas en Tinamït:

- **SAHYSMOD**: Modelo de salinidad de suelos. Ver su documentación. Para uso con Tinamït deberás descargar una versión especial modificada para funcionar desde la línea de comanda aquí (quieres el `SahysModConsole.exe`).

Envolturas planeadas:

- **PCSE**: Modelo de cultivos en puro Python, ver su documentación.
- **DSSAT**: Modelo de cultivos bien popular. Descarga grátis.
- **SWAT+**: Modelo hidrológico, gratis y de fuente abierta. Descarga aquí.

2.16 Interfaz de Programación

2.16.1 Modelos BF

Autogeneración

Modelos disponibles

class tinamit.envolt.bf.**ModeloBF**(*variables*, *nombre='bf'*)

La clase pariente para todos modelos biofísicos.

classmethod prb_egreso()

Debe devolver la ubicación de un archivo de egresos y una función que lo puede leer.

Returns

Return type tuple[str, Callable]

classmethod prb_ingreso()

Debe devolver la ubicación de un archivo de ingresos y una función que lo puede leer.

Returns

Return type tuple[str, Callable]

classmethod prb_simul()
Debe devolver la ubicación de un archivo de ingresos para correr una simulación de prueba.

Returns

Return type str

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.bf.**ModeloDeterminado**(*tmñ_ciclo*, *variables*, *nombre='bf'*)
La clase pariente para todos modelos que correr por un número predeterminado de pasos a cada simulación.

avanzar_modelo(*n_ciclos*)
Avanzar el modelo por un número determinado de ciclos.

Parameters **n_ciclos** (*int*) – El número de ciclos que hay que avanzar.

incrementar(*rebanada*)
Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.bf.**ModeloBloques**(*variables*, *nombre='bf'*)

Parameters

- **variables** (VariablesModBloques) –
- **nombre** (*str*) –

avanzar_modelo(*n_ciclos*)
Avanzar el modelo por un número determinado de ciclos.

Parameters **n_ciclos** (*int*) – El número de ciclos que hay que avanzar.

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.bf.**ModeloIndeterminado**(*variables*, *nombre='bf'*)
La clase pariente para todos modelos que avanzan por un número indeterminado de pasos a cada corrida.

incrementar(*rebanada*)
Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.bf.**ModeloImpaciente**(*tmñ_ciclo*, *variables*, *nombre='bf'*)
La clase pariente para modelos que deben correr por varios pasos al mismo tiempo, es decir, `Indeterminado` y `Determinado`.

iniciar_modelo(*corrida*)
Inicia la simulación. En general no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **corrida** (*Corrida*) – La corrida.

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

2.16.2 Modelos DS

Autogeneración

exception tinamit.envolt.mds._auto.**ErrorNoInstalado**
Error para devolver si no está instalada una envoltura.

tinamit.envolt.mds._auto.**gen_mds**(*archivo*)
Automáticamente generar un `ModeloDS` desde un archivo.

Parameters **archivo** (*str*) –

Returns

Return type *ModeloDS*

tinamit.envolt.mds._auto.**olvidar_envolt_mds**(*envoltura*)
Borra una envoltura del registro global.

Parameters **envoltura** – La envoltura que ya no quieres.

tinamit.envolt.mds._auto.**registrar_envolt_mds**(*envoltura*)
Registra una nueva envoltura en Tinamït.

Parameters **envoltura** – La nueva envoltura.

Modelos disponibles

class tinamit.envolt.mds.**ModeloDS**(*variables*, *nombre='mds'*)

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.mds.pysd.**ModeloPySD**(*archivo*, *nombre='mds'*)

Envoltura para modelos PySD.

cambiar_vals(*valores*)

Esta función cambia el valor de uno o más variables del modelo.

Parameters **valores** (*dict*) – Un diccionario de variables y sus valores para cambiar.

incrementar(*rebanada*)

Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

iniciar_modelo(*corrida*)

Inicia la simulación. En general no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **corrida** (*Corrida*) – La corrida.

paralelizable()

Indica si el modelo actual se puede paralelizar de manera segura o no. Si implementas una subclase paralelizable, reimplementar esta función para devolver `True`.

¿No sabes si es paralelizable tu modelo?

Respuesta larga: Si el modelo se puede paralelizar (con corridas de nombres distintos) sin encontrar dificultades técnicas (sin riesgo que las corridas paralelas terminen escribiendo en los mismos archivos de egreso), entonces sí es paralelizable tu modelo.

Respuesta rápida: 95% seguro que sí.

Returns Si el modelo es paralelizable o no.

Return type bool

unidad_tiempo()

Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

class tinamit.envolt.mds.vensim_dll.**ModeloVensimDLL**(*archivo*, *nombre='mds'*)

Esta es la envoltura para modelos de tipo Vensim. Puede leer y controlar cualquier modelo Vensim para que se pueda emplear en Tinamït. Necesitarás la versión DSS de Vensim para que funcione.

cambiar_vals(*valores*)

Esta función cambia el valor de uno o más variables del modelo.

Parameters **valores** (*dict*) – Un diccionario de variables y sus valores para cambiar.

cerrar()

Cierre la simulación Vensim.

incrementar(*rebanada*)

Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

iniciar_modelo(*corrida*)
Inicia la simulación. En general no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **corrida** (*Corrida*) – La corrida.

classmethod instalado()
Si tu modelo depiende en una instalación de otro programa externo a Tinamït, puedes reimplementar esta función para devolver `True` si el modelo está instalado y `False` sino.

Returns Si el modelo está instalado completamente o no.

Return type bool

paralelizable()
Indica si el modelo actual se puede paralelizar de manera segura o no. Si implementas una subclase paralelizable, reimplementar esta función para devolver `True`.

¿No sabes si es paralelizable tu modelo?

Respuesta larga: Si el modelo se puede paralelizar (con corridas de nombres distintos) sin encontrar dificultades técnicas (sin riesgo que las corridas paralelas terminen escribiendo en los mismos archivos de egreso), entonces sí es paralelizable tu modelo.

Respuesta rápida: 95% seguro que sí.

Returns Si el modelo es paralelizable o no.

Return type bool

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

Variables

class tinamit.envolt.mds.**VarAuxiliar**(*nombre*, *unid*, *ec*, *parientes*, *inic*, *subs=None*, *líms=None*, *info=''*)
Un variable auxiliar.

class tinamit.envolt.mds.**VarConstante**(*nombre*, *unid*, *ec*, *parientes*, *inic*, *subs=None*, *líms=None*, *info=''*)
Un variable constante.

class tinamit.envolt.mds.**VarInic**(*nombre*, *unid*, *ec*, *parientes*, *inic*, *subs=None*, *líms=None*, *info=''*)
Un variable inicial.

class tinamit.envolt.mds.**VarNivel**(*nombre*, *unid*, *ec*, *parientes*, *inic*, *subs=None*, *líms=None*, *info=''*)
Un variable nivel.

class tinamit.envolt.mds.**VarMDS**(*nombre*, *unid*, *ingr*, *egr*, *ec*, *parientes*, *inic*, *subs=None*, *líms=None*, *info=''*)
Un variable de un modelo `ModeloDS`.

class tinamit.envolt.mds.**VariablesMDS**(*variables*)
Representa los variables de un modelo `ModeloDS`.

2.16.3 Conectado

class tinamit.conect.**Conectado**(*bf*, *mds*, *nombre='conectado'*)
Un modelo que conecta un *ModeloDS* con un ModeloBF.

conectar(*var_mds*, *var_bf*, *mds_fuente*, *conv=None*)
Una función para conectar variables entre el modelo biofísico y el modelo DS.

Parameters

- **var_mds** (*str*) – El nombre del variable en el modelo DS.
- **var_bf** (*str*) – El nombre del variable correspondiente en el modelo biofísico.
- **mds_fuente** (*bool*) – Si True, el modelo DS es el modelo fuente para la conexión. Sino, será el modelo biofísico.
- **conv** (*float*) – El factor de conversión entre los variables.

desconectar(*var_mds*)
Esta función deshacer una conexión entre el modelo biofísico y el modelo DS. Se especifica la conexión por el nombre del variable en el modelo DS.

Parameters **var_mds** (*str*) – El nombre del variable conectado en el modelo DS.

class tinamit.conect.**SuperConectado**(*modelos*, *nombre='SuperConectado'*)
Esta clase representa el más alto nivel posible de modelo conectado. Tiene la función muy útil de poder conectar instancias de sí misma, así permitiendo la conexión de números arbitrarios de modelos anidados.

cambiar_vals(*valores*)
Esta función cambia el valor de uno o más variables del modelo.

Parameters **valores** (*dict*) – Un diccionario de variables y sus valores para cambiar.

cerrar()
Esta función toma acciones necesarias para terminar la simulación y cerrar el modelo, si aplica.

incrementar(*rebanada*)
Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al super cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

iniciar_modelo(*corrida*)
Inicia la simulación. En general no llamarías esta función directamente.

No se te olvide una llamada al super cuando reimplementas esta función.

Parameters **corrida** (*Corrida*) – La corrida.

paralelizable()
Indica si el modelo actual se puede paralelizar de manera segura o no. Si implementas una subclase paralelizable, reimplementar esta función para devolver True.

¿No sabes si es paralelizable tu modelo?

Respuesta larga: Si el modelo se puede paralelizar (con corridas de nombres distintos) sin encontrar dificultades técnicas (sin riesgo que las corridas paralelas terminen escribiendo en los mismos archivos de egreso), entonces sí es paralelizable tu modelo.

Respuesta rápida: 95% seguro que sí.

Returns Si el modelo es paralelizable o no.

Return type bool

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

2.16.4 Modelo

class tinamit.mod.modelo.**Modelo**(*variables*, *nombre*)
Todas las cosas en Tinamit son instancias de *Modelo*, que sea un modelo de dinámicas de los sistemas, un modelo de cultivos o de suelos o de clima, o un modelo conectado. Cada tipo de modelo se representa por subclases específicas. Por eso, la gran mayoría de los métodos definidos aquí se implementan de manera independiente en cada subclase de *Modelo*.

La función de inicialización de todos modelos, conectados o no.

Parameters

- **variables** (VariablesMod) – Los variables del modelo.
- **nombre** (*str*) – El nombre del modelo.

cambiar_vals(*valores*)
Esta función cambia el valor de uno o más variables del modelo.

Parameters **valores** (*dict*) – Un diccionario de variables y sus valores para cambiar.

cerrar()
Esta función toma acciones necesarias para terminar la simulación y cerrar el modelo, si aplica.

conectar_var_clima(*var*, *var_clima*, *conv*, *combin='prom'*)
Conecta un variable climático.

Parameters

- **var** (*str*) – El nombre interno del variable en el modelo.
- **var_clima** (*str*) – El nombre oficial del variable climático.
- **conv** (*number*) – La conversión entre el variable clima en Tinamït y el variable correspondiente en el modelo.
- **combin** (*str or function or None*) – Si este variable se debe adicionar o tomar el promedio entre varios pasos. Puede ser `prom`, `total`, o una función. Si es `None`, se tomará el último día en el caso de pasos de más de 1 día.

correr()
Efectuar una simulación ya inicializada. En general, no llamarías esta función directamente.

classmethod estab_conf(*llave*, *valor*)
Establece un valor de configuración.

Parameters

- **llave** (*str*) – El parámetro de configuración.
- **valor** (*str or int or float or list or bool or dict*) – El valor del parámetro.

incrementar(*rebanada*)
Incrementa el modelo. En general, no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **rebanada** (*Rebanada*) – La rebanada del incremento.

iniciar_modelo(*corrida*)
Inicia la simulación. En general no llamarías esta función directamente.

No se te olvide una llamada al `super` cuando reimplementas esta función.

Parameters **corrida** (*Corrida*) – La corrida.

classmethod instalado()
Si tu modelo depiende en una instalación de otro programa externo a Tinamït, puedes reimplementar esta función para devolver `True` si el modelo está instalado y `False` sino.

Returns Si el modelo está instalado completamente o no.

Return type bool

classmethod obt_conf(*llave*, *auto=None*, *cond=None*, *mnsj_err=None*)
Obtiene un valor de configuración de la subclase de modelo.

Parameters

- **llave** (*str*) – El parámetro de configuración.
- **auto** (*str or int or float or list or bool or dict*) – Un valor automático a aplicar si no se encuentra en el diccionario de configuración.
- **cond** – Una condición para validar el valor; si no pasa la condición, se tratará como valor que falta.
- **mnsj_err** – Un mensaje de aviso para devolver al usuario si no se encuentra el valor.

Returns El valor de configuración

Return type str, int, float, list, bool, dict

paralelizable()
Indica si el modelo actual se puede paralelizar de manera segura o no. Si implementas una subclase paralelizable, reimplementar esta función para devolver `True`.

¿No sabes si es paralelizable tu modelo?

Respuesta larga: Si el modelo se puede paralelizar (con corridas de nombres distintos) sin encontrar dificultades técnicas (sin riesgo que las corridas paralelas terminen escribiendo en los mismos archivos de egreso), entonces sí es paralelizable tu modelo.

Respuesta rápida: 95% seguro que sí.

Returns Si el modelo es paralelizable o no.

Return type bool

simular(*t*, *nombre='Tinamït'*, *extern=None*, *clima=None*, *vars_interés=None*)

Parameters

- **t** (*int or EspecTiempo*) – La especificación del eje de tiempo. Si es `int`, significará el número de pasos.
- **nombre** (*str*) – El nombre de la corrida.
- **extern** (*Extern or pd.DataFrame or xr.Dataset or dict*) – Valores externos para la simulación.
- **clima** (*Clima*) – El clima de la simulación.
- **vars_interés** (*list*) – Los variables para incluir en los resultados

Returns

Return type *ResultadosSimul*

simular_grupo(*ops_grupo*, *nombre='Tinamït'*, *paralelo=False*)
Efectua un grupo de simulaciones. Muy útil para accelerar corridas múltiples.

Parameters

- **ops_grupo** (*PlantillaOpsSimulGrupo*) – Las opciones de simulación en grupo.
- **nombre** (*str*) – El nombre de la simulación.
- **paralelo** (*bool*) – Si se simula en paralelo o no. Si el modelo no soporte corridas en paralelo, se ignorará este argumento.

Returns

Return type *ResultadosGrupo*

unidad_tiempo()
Esta función debe devolver la unidad de tiempo empleada por el modelo.

Returns La unidad de tiempo (p. ej., 'meses', '', etc.)

Return type str

2.16.5 Pruebas

tinamit.mod.prbs.**verificar_leer_egr**(*caso*, *cls*)
Verifica que una envoltura lee bien sus datos de egreso.

Parameters

- **caso** (*unittest.TestCase*) – El caso de prueba.
- **cls** – La clase del modelo para comprobar.

tinamit.mod.prbs.**verificar_leer_ingr**(*caso*, *cls*)
Verifica que una envoltura lee bien sus datos de ingreso.

Parameters

- **caso** (*unittest.TestCase*) – El caso de prueba.
- **cls** – La clase del modelo para comprobar.

tinamit.mod.prbs.**verificar_simul**(*caso*, *cls*)
Verifica que una envoltura simula bien. No correrá si no está instalada la envoltura.

Parameters

- **caso** (*unittest.TestCase*) – El caso de prueba.
- **cls** – La clase del modelo para comprobar.

2.16.6 Geografía

class tinamit.geog.región.**Lugar**(*nombre*, *nivel*, *cód=None*, *sub_lugares=None*)
Un lugar dado en una geografía.

Parameters

- **nombre** (*str*) – El nombre del lugar.

- **nivel** (Nivel) – El nivel geográfico correspondiente.
- **cód** (*str*) – El identificador único de este lugar. Si es None, se tomará su nombre como identificador.
- **sub_lugares** – Lugares que se encuentre adentro de este.

buscar_nombre(*nombre*, *nivel=None*)
Devuelve el sublugar con el nombre dado.

Parameters

- **nombre** (*str*) – El nombre del lugar deseado.
- **nivel** (*Nivel or str*) – Desambiguación en el caso que hayan múltiples lugares con el mismo nombre en distintos niveles.

Returns

Return type *Lugar*

hijos_inmediatos(*ord_niveles=None*)
Devuelve los hijos inmediatos de este Lugar.

Parameters **ord_niveles** (*list*) – Desambiguación para lugares con niveles paralelos.

Returns

Return type list[*Lugar*]

lugares(*en=None*, *nivel=None*)
Devolver los sublugares presentes en este lugar.

Parameters

- **en** (*str or Lugar*) – Sublugar al cual limitir la búsqueda.
- **nivel** (*Nivel or str or list*) – Opción para limitir los resultados a uno o más niveles.

Returns

Return type set[*Lugar*]

pariente(*lugar*, *ord_niveles=None*, *todos=False*)
Obtener el pariente de un sublugar dado.

Parameters

- **lugar** (*str or Lugar*) – Un sublugar cuyo pariente queremos.
- **ord_niveles** (*list*) – Desambiguación para lugares con niveles paralelos.
- **todos** (*bool*) – Si queremos todos los parientes del lugar, o solamente el más cercaco.

Returns

Return type *Lugar*

class tinamit.geog.región.**Nivel**(*nombre*, *subniveles=None*)
Un nivel geográfico (p. ej, municipio o departamento.

Parameters

- **nombre** (*str*) – El nombre del nivel.
- **subniveles** (*list of Nivel*) – Lista de subniveles.

tinamit.geog.región.**gen_lugares**(*archivo*, *nivel_base*, *nombre=None*, *col_cód='Código'*)
Genera un lugar con todos los niveles y sublugares asociados desde un archivo `.csv`.

Cada columna en el `.csv` debe empezar con el nombre de un nivel, con la excepción de la columna `col_cód`, la cual tendrá el código identificador único de cada lugar.

Cada fila representa un lugar, con su **nombre** en la columna correspondiendo al nivel de este lugar y el **código** del lugar pariente en las otras columnas. Si un nivel no se aplica a un lugar (por ejemplo, un departamento no tendrá municipio pariente), se deja vacía la célula.

Parameters

- **archivo** (*str*) – El archivo `.csv`.
- **nivel_base** (*str*) – El el nivel más alto. Por ejemplo, si tu csv entero representa un país, sería `país`.
- **nombre** (*str*) – El nombre del lugar correspondiendo al nivel más alto. Por ejemplo, "Guatemala".
- **col_cód** (*str*) – El nombre de la columna con los códigos de cada sublugar.

Returns

Return type *Lugar*

2.16.7 Mapas

class tinamit.geog.mapa.**Agua**(*archivo*, *llenar=True*)
Representa áreas de agua.

Parameters

- **archivo** (*str*) – El archivo `.shp`.
- **llenar** (*bool*) – Si hay que llenar el cuerpo de agua o no.

class tinamit.geog.mapa.**Bosque**(*archivo*)
Representa áreas con bosque.

class tinamit.geog.mapa.**Calle**(*archivo*)
Representa calles.

class tinamit.geog.mapa.**Ciudad**(*archivo*)
Representa áreas urbanas.

class tinamit.geog.mapa.**Forma**(*archivo*, *llenar*, *alpha*)
Clase pariente para todas las formas que se pueden dibujar.

dibujar(*ejes*, *fig*)
Agrega la forma a la figura.

Parameters

- **ejes** – Los ejes de la figura.
- **fig** – La figura.

class tinamit.geog.mapa.**FormaDinámica**(*archivo*, *escala_colores=None*, *llenar=True*, *alpha=1*)
Forma cuyos colores se asignan según valores numéricos.

Parameters

- **archivo** (*str*) – El archivo `.shp`.

- **escala_colores** (*list or tuple or str or int or None*) – Lista de dos colores para establecer una escala de colores. Si es un solo color, se agregará el color blanco. Si es `-1`, se inverserán los colores automáticos.
- **llenar** (*bool*) – Si hay que llenar la forma o simplement delinear su contorno.
- **alpha** (*float or int*) – La opacidad del interior de la forma. Solamente aplica si `llenar` est `False`.

dibujar(*ejes*, *fig*)
Agrega la forma a la figura.

Parameters

- **ejes** – Los ejes de la figura.
- **fig** – La figura.

estab_valores(*valores*, *escala_valores=None*, *unidades=None*)
Establece los valores para colorar.

Parameters

- **valores** (*np.ndarray or dict*) – Los valores para dibujar. Debe ser del mismo tamaño que el archivo `.shp` en `archivo`.
- **escala_valores** (*tuple or list or None*) – La escala para el rango de colores. Si es `None`, se ajustará el rango según de los valores dados.
- **unidades** (*str, optional*) – Las unidades.

class tinamit.geog.mapa.**FormaDinámicaNombrada**(*archivo*, *col_id*, *escala_colores=None*, *llenar=True*, *alpha=1*)
Forma dinámica cuyos valores se asignan a los polígonos de la forma `.shp` por su llave en el diccionario de valores.

Parameters

- **archivo** (*str*) – La archivo `.shp`.
- **col_id** (*str*) – La columna en el archivo `.shp` con el nomrbe de cada polígono.
- **escala_colores** (*list or tuple or str or int or None*) – Lista de dos colores para establecer una escala de colores. Si es un solo color, se agregará el color blanco. Si es `-1`, se inverserán los colores automáticos.
- **llenar** (*bool*) – Si hay que llenar la forma o simplement delinear su contorno.
- **alpha** (*float or int*) – La opacidad del interior de la forma. Solamente aplica si `llenar` est `False`.

class tinamit.geog.mapa.**FormaDinámicaNumérica**(*archivo*, *col_id=None*, *escala_colores=None*, *llenar=True*, *alpha=1*)
Forma dinámica cuyos valores se asignan a los polígonos de la forma `.shp` por su orden en la matriz de valores.

Parameters

- **archivo** (*str*) – El archivo `.shp`.
- **col_id** (*str, optional*) – La columna con el número de cada polígono en la forma `.shp`. Si es `None`, se asiñará número según su orden en la forma `.shp`.
- **escala_colores** (*list or tuple or str or int or None*) – Lista de dos colores para establecer una escala de colores. Si es un solo color, se agregará el color blanco. Si es `-1`, se inverserán los colores automáticos.

- **llenar** (*bool*) – Si hay que llenar la forma o simplement delinear su contorno.
- **alpha** (*float or int*) – La opacidad del interior de la forma. Solamente aplica si `llenar` est `False`.

class tinamit.geog.mapa.**FormaEstática**(*archivo*, *color*, *llenar*, *alpha*)
Clase de base para formas estáticas en el mapa, cuyos colores no cambian.

dibujar(*ejes*, *fig*)
Agrega la forma a la figura.

Parameters

- **ejes** – Los ejes de la figura.
- **fig** – La figura.

class tinamit.geog.mapa.**OtraForma**(*archivo*)
Representa otras áreas no representadas por las otras formas disonibles.

tinamit.geog.mapa.**dibujar_mapa**(*formas*, *archivo=None*, *título=None*, *fig=None*)
Dibuja un mapa.

Parameters

- **formas** (*list of Forma*) – Las formas para incluir.
- **archivo** (*str*) – Dónde hay que guardar el gráfico. Si es `None`, no se guardará el gráfico.
- **título** (*str*) – El título del mapa.
- **fig** (*matplotlib.Figure*) – Figura para dibujar el mapa.

Returns La figura y sus ejes.

Return type tuple[Figure, Axes]

tinamit.geog.mapa.**dibujar_mapa_de_res**(*forma_dinámica*, *res*, *var*, *t*, *escala=None*, *título=''*, *directorio=None*, *otras_formas=None*)
Dibujar los resultados de una simulación en un mapa.

Parameters

- **forma_dinámica** (FormaDinámica) – La forma cuyos colores variarán según los resultados.
- **res** (ResultadosSimul *or* ResultadosGrupo) – Los resultados para dibujar.
- **var** (*str*) – El variable de interés.
- **t** (*int or tuple or range or list*) – Los tiempos a los cuales queremos graficar los resultados.
- **escala** (*tuple*) – El rango para aplicar colores. Si es `None`, se aplicará según los datos de los resultados.
- **título** (*str*) – El título del gráfico.
- **directorio** (*str*) – Dónnde hay que guardar el gráfico.
- **otras_formas** (*list of FormaEstática or* FormaEstática) – Las otras formas (estáticas) para incluir en el gráfico.

2.16.8 Datos

class tinamit.datos.fuente.**Fuente**(*nombre*, *variables*, *lugares=None*, *fechas=None*)
La clase pariente para fuentes de datos.

class tinamit.datos.fuente.**FuenteBaseXarray**(*obj*, *nombre*, *lugares=None*, *fechas=None*)
Fuente para datos en formato de `Dataset` de `xarray`.

Parameters

- **obj** (*xarray.Dataset*) – Los datos
- **nombre** (*str*) – El nombre de la fuente.
- **lugares** (*str or np.ndarray or list*) – Los lugares que corresponden a los datos. Puede se nombre de una columna en el `Dataset`, el nombre de un lugar de cual vienen todos los datos, o una lista de los lugares.
- **fechas** (*str or np.ndarray or list or datetime.datetime*) – Las fechas de los datos.

class tinamit.datos.fuente.**FuenteCSV**(*archivo*, *nombre=None*, *lugares=None*, *fechas=None*, *cód_vacío=None*)
Fuente para archivos `.csv`.

Parameters

- **archivo** (*str*) – El archivo con los datos.
- **nombre** (*str*) – El nombre de la fuente.
- **lugares** (*str or np.ndarray or list*) – Los lugares que corresponden a los datos. Puede se nombre de una columna en el csv, el nombre de un lugar de cual vienen todos los datos, o una lista de los lugares.
- **fechas** (*str or np.ndarray or list or datetime.datetime*) – Las fechas de los datos.
- **cód_vacío** – Código para identificar variables que faltan. `NA` y `NaN` ya están reconocidos.

class tinamit.datos.fuente.**FuenteDic**(*dic*, *nombre*, *lugares=None*, *fechas=None*)
Fuente de datos en forma de diccionario.

Parameters

- **dic** (*dict*) – El diccionario con los datos.
- **nombre** (*str*) – El nombre de la fuente.
- **lugares** (*str or np.ndarray or list*) – Los lugares que corresponden a los datos. Puede se nombre de una llave en el dictionario, el nombre de un lugar de cual vienen todos los datos, o una lista de los lugares.
- **fechas** (*str or np.ndarray or list or datetime.datetime*) – Las fechas de los datos.

class tinamit.datos.fuente.**FuentePandas**(*obj*, *nombre*, *lugares=None*, *fechas=None*)
Fuente para datos en formato de `DataFrame` de `xarray`.

Parameters

- **obj** (*pd.DataFrame*) – Los datos
- **nombre** (*str*) – El nombre de la fuente.

- **lugares** (*str or np.ndarray or list*) – Los lugares que corresponden a los datos. Puede se nombre de una columna en el `Dataset`, el nombre de un lugar de cual vienen todos los datos, o una lista de los lugares.
- **fechas** (*str or np.ndarray or list or datetime.datetime*) – Las fechas de los datos.

class tinamit.datos.fuente.**FuenteVarXarray**(*obj*, *nombre*, *lugares=None*, *fechas=None*)

Fuente para datos en formato de `DataArray` de `xarray`.

Parameters

- **obj** (*xarray.DataArray*) – Los datos
- **nombre** (*str*) – El nombre de la fuente.
- **lugares** (*str or np.ndarray or list*) – Los lugares que corresponden a los datos. Puede se nombre de una columna en el `DataArray`, el nombre de un lugar de cual vienen todos los datos, o una lista de los lugares.
- **fechas** (*str or np.ndarray or list or datetime.datetime*) – Las fechas de los datos.

class tinamit.datos.bd.**BD**(*fuentes*)

Una base de datos combina varias *Fuente*.

Parameters **fuentes** (*Fuente or list*) – Las fuentes de la base de datos.

interpolar(*vars_interés*, *lugares=None*, *fechas=None*, *extrap=False*)

Interpola datos por fecha, tomando el lugar en cuenta.

Parameters

- **vars_interés** (*str or list*) – Los variables de interés.
- **lugares** (*list*) – Lugares de interés.
- **fechas** (*list or str or datetime.datetime*) – Las fechas de interés.
- **extrap** (*bool*) – Si hay que extrapolar también.

Returns `xr.DataArray` si `vars_interés` es `str`, `xr.Dataset` si `vars_interés` es `list`.

Return type xr.DataArray, xr.Dataset

obt_vals(*vars_interés=None*, *lugares=None*, *fechas=None*)

Devuelve los valores de unos variables de interés.

Parameters

- **vars_interés** (*str or list*) – Los variables de interés.
- **lugares** (*list*) – Lugares de interés.
- **fechas** (*tuple or list*) –

Returns `xr.DataArray` si `vars_interés` es `str`, `xr.Dataset` si `vars_interés` es `list`.

Return type xr.DataArray, xr.Dataset

2.16.9 Calibraciones

Ecuaciones

class tinamit.calibs.ec.**CalibradorEc**(*ec*, *paráms*, *nombre=None*, *dialecto='tinamït'*)
Clase pariente para implementaciones de calibradores de ecuaciones.

Parameters

- **ec** (*str or Ec*) – La ecuación para calibrar.
- **paráms** (*list*) – La lista de nombres de parámetros en la ecuación (variables que hay que calibrar).
- **nombre** (*str*) – El nombre del variable dependiente en la ecuación. Obligatorio si la ecuación no especifica variable independiente sí misma (p. ej., `x * b + a` en vez de `y = x * b + a`.
- **dialecto** (*str*) – El dialecto de la ecuación. Puede ser `tinamït` o `vensim`.

calibrar(*bd*, *lugar=None*, *líms_paráms=None*, *ops=None*, *corresp_vars=None*, *ord_niveles=None*)
Efectua la calibración.

Parameters

- **bd** (BD) – La base de datos con observaciones para los variables en la ecuación.
- **lugar** (Lugar) – El lugar cuyos sublugares hay que calibrar; si es `None` se calibrará la ecuación con todos los datos en `bd` sin tener su lugar en cuenta.
- **líms_paráms** (*list*) – Límites teoréticos para los parámetros.
- **ops** (*dict*) – Opciones que se pasarán directamente a la función de calibración.
- **corresp_vars** (*dict*) – Diccionario de correspondencia entre los nombres de los variables en `bd` y sus nombres en la ecuación.
- **ord_niveles** (*list*) – Desambiguación del orden de niveles.

Returns Diccionario con las calibraciones de cada lugar.

Return type dict

class tinamit.calibs.ec.**CalibradorEcOpt**(*ec*, *paráms*, *nombre=None*, *dialecto='tinamït'*)
Calibrador de ecuaciones con algoritmo de optimización.

calibrar(*bd*, *lugar=None*, *líms_paráms=None*, *ops=None*, *corresp_vars=None*, *ord_niveles=None*)
Efectua una calibración para cada lugar en `Lugar` según los datos en `bd`.

Parameters

- **bd** (BD) – La base de datos con observaciones para los variables en la ecuación.
- **lugar** (Lugar) – El lugar cuyos sublugares hay que calibrar; si es `None` se calibrará la ecuación con todos los datos en `bd` sin tener su lugar en cuenta.
- **líms_paráms** (*list*) – Límites teoréticos para los parámetros.
- **ops** (*dict*) – Opciones que se pasarán directamente a la función de calibración.
- **corresp_vars** (*dict*) – Diccionario de correspondencia entre los nombres de los variables en `bd` y sus nombres en la ecuación.
- **ord_niveles** (*list*) – Desambiguación del orden de niveles.

Returns Diccionario con las calibraciones de cada lugar.

Return type dict

class tinamit.calibs.ec.**CalibradorEcBayes**(*ec*, *paráms*, *nombre=None*, *dialecto='tinamït'*)
Calibrador de ecuaciones con inferencia bayesiana.

Parameters

- **ec** (*str or Ec*) – La ecuación para calibrar.
- **paráms** (*list*) – La lista de nombres de parámetros en la ecuación (variables que hay que calibrar).
- **nombre** (*str*) – El nombre del variable dependiente en la ecuación. Obligatorio si la ecuación no especifica variable independiente sí misma (p. ej., `x * b + a` en vez de `y = x * b + a`.
- **dialecto** (*str*) – El dialecto de la ecuación. Puede ser `tinamït` o `vensim`.

calibrar(*bd*, *lugar=None*, *líms_paráms=None*, *ops=None*, *corresp_vars=None*, *ord_niveles=None*, *jerárquico=True*)
Efectua una calibración bayesiana para cada lugar en `Lugar` según los datos en `bd`.

Parameters

- **bd** (`BD`) – La base de datos con observaciones para los variables en la ecuación.
- **lugar** (`Lugar`) – El lugar cuyos sublugares hay que calibrar; si es `None` se calibrará la ecuación con todos los datos en `bd` sin tener su lugar en cuenta.
- **líms_paráms** (*list*) – Límites teoréticos para los parámetros.
- **ops** (*dict*) – Opciones que se pasarán directamente a la función de calibración.
- **corresp_vars** (*dict*) – Diccionario de correspondencia entre los nombres de los variables en `bd` y sus nombres en la ecuación.
- **ord_niveles** (*list*) – Desambiguación del orden de niveles.
- **jerárquico** (*bool*) – Si empleamos inferencia bayesiana jerárquica o normal.

Returns Diccionario con las calibraciones de cada lugar.

Return type dict

Modelos

class tinamit.calibs.geog_mod.**SimuladorGeog**(*mod*)
Simulador geográfico.

Parameters **mod** (`Modelo`) – El modelo para simular.

simular(*t*, *vals_geog*, *vals_const=None*, *vars_interés=None*, *paralelo=False*)
Efectua una simulación geográfica.

Parameters

- **t** (*int or EspecTiempo*) – El eje de tiempo para la simulación.
- **vals_geog** (*dict*) – Diccionario de cada lugar con sus valores de parámetros.
- **vals_const** (*dict*) – Valores de parámetros cuyos valores no cambian según el lugar.
- **vars_interés** (*str or Variable or list*) – Los variables cuyos resultados nos interesan.
- **paralelo** (*bool*) – Si se puede simular en paralelo.

Returns

Return type *ResultadosGrupo*

class tinamit.calibs.geog_mod.**CalibradorGeog**(*mod*, *calibrador=<class 'tinamit.calibs.mod.CalibradorModSpotPy'>*)

Objeto para efectuar calibraciones geográficas.

Parameters

- **mod** (Modelo) – El modelo para calibrar.
- **calibrador** (*type*) – Una subclase de CalibradorMod.

class tinamit.calibs.geog_mod.**ValidadorGeog**(*mod*)

Objeto para correr validaciones de calibraciones geográficas.

Parameters **mod** (Modelo) – El modelo para validar.

validar(*t*, *datos*, *paráms=None*, *funcs=None*, *vars_extern=None*, *corresp_vars=None*)

Efectuar la validación.

Parameters

- **t** (*int or EspecTiempo*) – La especificación de tiempo para la validación.
- **datos** (BD) – La base de datos para la validación.
- **paráms** (*dict*) – Diccionario de los parámetros calibrados para cada lugar.
- **funcs** (*list*) – Funciones de validación para aplicar a los resultados.
- **vars_extern** (*str or list or Variable*) – Variable(s) exógenos cuyos valores se tomarán de la base de datos para alimentar la simulación y con los cuales por supuesto no se validará el modelo.
- **corresp_vars** – Diccionario de correspondencia entre nombres de valores en el modelo y en la base de datos.

Returns Diccionario de la validación del modelo para cada variable con datos en cada lugar.

Return type dict

class tinamit.calibs.valid.**ValidadorMod**(*mod*)

Clase para efectuar validaciones de un modelo.

Parameters **mod** (Modelo) – El modelo para validar.

validar(*t*, *datos*, *paráms=None*, *funcs=None*, *vars_extern=None*, *corresp_vars=None*)

Efectua la validación.

Parameters

- **t** (*int or EspecTiempo*) – La especificación de tiempo para la validación.
- **datos** (*xr.Dataset or xr.DataArray or str or pd.DataFrame or dict or Fuente or list*) – La base de datos para la validación.
- **paráms** (*dict*) – Diccionario de los parámetros calibrados para cada lugar.
- **funcs** (*list*) – Funciones de validación para aplicar a los resultados.
- **vars_extern** (*str or list or Variable*) – Variable(s) exógenos cuyos valores se tomarán de la base de datos para alimentar la simulación y con los cuales por supuesto no se validará el modelo.
- **corresp_vars** – Diccionario de correspondencia entre nombres de valores en el modelo y en la base de datos.

Returns Validación por variable.

Return type dict

2.16.10 Variables

class tinamit.mod.var.**Variable**(*nombre*, *unid*, *ingr*, *egr*, *inic=0*, *líms=None*, *info=''*)
La clase más general para variables de modelos en Tinamït.

Parameters

- **nombre** (*str*) – El nombre del variable.
- **unid** (*str or None*) – Las unidades del variable.
- **ingr** (*bool*) – Si es un ingreso al modelo.
- **egr** (*bool*) – Si es un egreso del modelo.
- **inic** (*int or float or np.ndarray*) – El valor inicial del modelo.
- **líms** (*tuple*) – Los límites del variable.
- **info** (*str*) – Descripción detallada del variable.

obt_val()
Devuelve el valor del variable.

poner_val(*val*)
Establece el valor del variable.

Parameters **val** (*int or float or np.ndarray*) – El nuevo valor.

reinic()
Reinicializa el variable a su valor pre-simulación.

class tinamit.mod.vars_mod.**VariablesMod**(*variables*)

Impacientes

class tinamit.envolt.bf.**VariablesModImpaciente**(*variables*)
Representa los variables de un modelo `Impaciente`.

act_paso(*paso*)
Actualizar el paso de los variables en el ciclo.

Parameters **paso** (*int*) – El paso actual en el ciclo.

vars_paso()
Devuelve los variables por paso.

Returns

Return type list

Determinados

class tinamit.envolt.bf.**VarPasoDeter**(*nombre*, *unid*, *ingr*, *egr*, *tmñ_ciclo*, *inic=0*, *líms=None*, *info=''*)
Un variable de un modelo `Determinado` que toma un valor distinto a cada paso (y no solamente a cada ciclo de simulación).

class tinamit.envolt.bf.**VariablesModDeter**(*variables*)
Representa los variables de un modelo `Determinado`.

Indeterminados

class tinamit.envolt.bf.**VarPasoIndeter**(*nombre*, *unid*, *ingr*, *egr*, *inic=0*, *líms=None*, *info=''*)
Representa un variable de un modelo `ModeloIndeterminado` cuyo valor cambia a cada paso (y no solamente a cada ciclo).

poner_vals_paso(*val*, *paso=None*)
Establece el valor del variable a un paso dado. Si `paso` es `None`, `val` debe ser una matriz donde eje 0 corresponde a todos los pasos del ciclo.

Parameters

- **val** (*np.ndarray*) – El nuevo valor.
- **paso** (*int*) – El paso al cual poner el nuevo valor del variable.

class tinamit.envolt.bf.**VariablesModIndeterminado**(*variables*)
Representa los variables de un modelo `ModeloIndeterminado`.

Bloques

class tinamit.envolt.bf.**VarBloque**(*nombre*, *unid*, *ingr*, *egr*, *tmñ_bloques*, *inic=0*, *líms=None*, *info=''*)

class tinamit.envolt.bf.**VariablesModBloques**(*variables*, *tmñ_bloques*)

act_paso(*paso*)
Actualizar el paso de los variables en el ciclo.

Parameters **paso** (*int*) – El paso actual en el ciclo.

2.16.11 Resultados

class tinamit.mod.res.**ResultadosGrupo**(*nombre*)
Resultados de una simulación por grupo.

class tinamit.mod.res.**ResultadosSimul**(*nombre*, *t*, *vars_interés*)
Resultados de una simulación.

a_dic()
Convierte los resultados en diccionario.

Returns

Return type dict

guardar(*frmt='json'*, *l_vars=None*)
Guarda los resultados en un archivo.

Parameters

- **frmt** (*str*) – El formato deseado. Puede ser `json` o `csv`.
- **l_vars** – La lista de variables de interés.

class tinamit.mod.res.**ResultadosVar**(*var*, *t*)
Los resultados de un variable.

2.16.12 Unidades

tinamit.unids.conv.**convertir**(*de*, *a*, *val=1*, *lengua=None*)
Esta función convierte un valor de una unidad a otra.

Parameters

- **de** (*str*) – La unidad original.
- **a** (*str*) – La unidad final.
- **val** (*float | int*) – El valor para convertir.
- **lengua** (*str*) – La lengua en la cual las unidades están especificadas.

Returns El valor convertido.

Return type float

tinamit.unids.conv.**definir_en_regu**(*unid*, *base*)
Esta funcion define una nueva unidad en el registro de unidades.

Parameters

- **unid** (*str*) – La unidad.
- **base** (*str*) – La dimensionalidad de la unidad para pasar a Pint.

tinamit.unids.trads.**act_arch_trads**(*l_d_t*)
Actualiza el fuente de traducciones.

Parameters **l_d_t** (*list[dict]*) –

tinamit.unids.trads.**agregar_sinónimos**(*unid*, *sinónimos*, *leng*, *guardar=False*)
Agrega sinónimos a una unidad.

Parameters

- **unid** (*str*) – La unidad original.
- **sinónimos** (*str | list*) – Los sinónimos.
- **leng** (*str*) – La lengua.
- **guardar** (*bool*) – Si guardamos los sinónimos para futuras sesiones de Python.

tinamit.unids.trads.**agregar_trad**(*unid*, *trad*, *leng_trad*, *leng_orig=None*, *guardar=True*)
Agregar una traducción a una unidad.

Parameters

- **unid** (*str*) – La unidad original.
- **trad** (*str*) – La traducción de la unidad.
- **leng_trad** (*str*) – La lengua de la traducción.
- **leng_orig** (*str*) – La lengua original.
- **guardar** (*bool*) – Si hay que guardar la traducción para futuras sesiones de Python.

tinamit.unids.trads.**buscar_singular**(*u*)
Busca las formas singulares posibles de una unidad.

Parameters **u** (*str*) – La unidad potencialmente plurial.

Returns Una lista de las formas singulares potenciales.

Return type list[str]

tinamit.unids.trads.**trad_unid**(*unid*, *leng_final*, *leng_orig=None*, *falla_silencio=True*)
Traduce una unidad sencilla (no compuesta).

Parameters

- **unid** (*str*) – La unidad para traducir.
- **leng_final** (*str*) – La lengua a la cual traducir.
- **leng_orig** (*str*) – La lengua original de la unidad. Si no se especifica, se intentará adivinarla.
- **falla_silencio** (*bool*) – Si hay que devolver un error si no se encontró traducción.

Returns La unidad traducida.

Return type str

Python Module Index

t

tinamit.datos.bd, 44
tinamit.datos.fuente, 43
tinamit.envolt.mds._auto, 32
tinamit.geog.mapa, 40
tinamit.geog.región, 38
tinamit.mod.prbs, 38
tinamit.mod.res, 49
tinamit.unids.conv, 50
tinamit.unids.trads, 50

Index

A

a_dic() (*tinamit.mod.res.ResultadosSimul method*), 49
act_arch_trads() (*in module tinamit.unids.trads*), 50
act_paso() (*tinamit.envolt.bf.VariablesModBloques method*), 49
act_paso() (*tinamit.envolt.bf.VariablesModImpaciente method*), 48
agregar_sinónimos() (*in module tinamit.unids.trads*), 50
agregar_trad() (*in module tinamit.unids.trads*), 50
Agua (*class in tinamit.geog.mapa*), 40
avanzar_modelo() (*tinamit.envolt.bf.ModeloBloques method*), 31
avanzar_modelo() (*tinamit.envolt.bf.ModeloDeterminado method*), 31

B

BD (*class in tinamit.datos.bd*), 44
Bosque (*class in tinamit.geog.mapa*), 40
buscar_nombre() (*tinamit.geog.región.Lugar method*), 39
buscar_singular() (*in module tinamit.unids.trads*), 50

C

CalibradorEc (*class in tinamit.calibs.ec*), 45
CalibradorEcBayes (*class in tinamit.calibs.ec*), 46
CalibradorEcOpt (*class in tinamit.calibs.ec*), 45
CalibradorGeog (*class in tinamit.calibs.geog_mod*), 47
calibrar() (*tinamit.calibs.ec.CalibradorEc method*), 45
calibrar() (*tinamit.calibs.ec.CalibradorEcBayes method*), 46
calibrar() (*tinamit.calibs.ec.CalibradorEcOpt method*), 45
Calle (*class in tinamit.geog.mapa*), 40
cambiar_vals() (*tinamit.conect.SuperConectado method*), 35
cambiar_vals() (*tinamit.envolt.mds.pysd.ModeloPySD method*), 33
cambiar_vals() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 33
cambiar_vals() (*tinamit.mod.modelo.Modelo method*), 36
cerrar() (*tinamit.conect.SuperConectado method*), 35
cerrar() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 33
cerrar() (*tinamit.mod.modelo.Modelo method*), 36
Ciudad (*class in tinamit.geog.mapa*), 40
Conectado (*class in tinamit.conect*), 35
conectar() (*tinamit.conect.Conectado method*), 35
conectar_var_clima() (*tinamit.mod.modelo.Modelo method*), 36
convertir() (*in module tinamit.unids.conv*), 50
correr() (*tinamit.mod.modelo.Modelo method*), 36

D

definir_en_regu() (*in module tinamit.unids.conv*), 50
desconectar() (*tinamit.conect.Conectado method*), 35
dibujar() (*tinamit.geog.mapa.Forma method*), 40
dibujar() (*tinamit.geog.mapa.FormaDinámica method*), 41
dibujar() (*tinamit.geog.mapa.FormaEstática method*), 42
dibujar_mapa() (*in module tinamit.geog.mapa*), 42
dibujar_mapa_de_res() (*in module tinamit.geog.mapa*), 42

E

ErrorNoInstalado, 32
estab_conf() (*tinamit.mod.modelo.Modelo class method*), 36

estab_valores() (*tinamit.geog.mapa.FormaDinámica method*), 41

F

Forma (*class in tinamit.geog.mapa*), 40
FormaDinámica (*class in tinamit.geog.mapa*), 40
FormaDinámicaNombrada (*class in tinamit.geog.mapa*), 41
FormaDinámicaNumérica (*class in tinamit.geog.mapa*), 41
FormaEstática (*class in tinamit.geog.mapa*), 42
Fuente (*class in tinamit.datos.fuente*), 43
FuenteBaseXarray (*class in tinamit.datos.fuente*), 43
FuenteCSV (*class in tinamit.datos.fuente*), 43
FuenteDic (*class in tinamit.datos.fuente*), 43
FuentePandas (*class in tinamit.datos.fuente*), 43
FuenteVarXarray (*class in tinamit.datos.fuente*), 44

G

gen_lugares() (*in module tinamit.geog.región*), 39
gen_mds() (*in module tinamit.envolt.mds._auto*), 32
guardar() (*tinamit.mod.res.ResultadosSimul method*), 49

H

hijos_inmediatos() (*tinamit.geog.región.Lugar method*), 39

I

incrementar() (*tinamit.conect.SuperConectado method*), 35
incrementar() (*tinamit.envolt.bf.ModeloDeterminado method*), 31
incrementar() (*tinamit.envolt.bf.ModeloIndeterminado method*), 31
incrementar() (*tinamit.envolt.mds.pysd.ModeloPySD method*), 33
incrementar() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 33
incrementar() (*tinamit.mod.modelo.Modelo method*), 36
iniciar_modelo() (*tinamit.conect.SuperConectado method*), 35
iniciar_modelo() (*tinamit.envolt.bf.ModeloImpaciente method*), 32
iniciar_modelo() (*tinamit.envolt.mds.pysd.ModeloPySD method*), 33
iniciar_modelo() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 33
iniciar_modelo() (*tinamit.mod.modelo.Modelo method*), 37
instalado() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL class method*), 34
instalado() (*tinamit.mod.modelo.Modelo class method*), 37
interpolar() (*tinamit.datos.bd.BD method*), 44

L

Lugar (*class in tinamit.geog.región*), 38
lugares() (*tinamit.geog.región.Lugar method*), 39

M

Modelo (*class in tinamit.mod.modelo*), 36
ModeloBF (*class in tinamit.envolt.bf*), 30
ModeloBloques (*class in tinamit.envolt.bf*), 31
ModeloDeterminado (*class in tinamit.envolt.bf*), 31
ModeloDS (*class in tinamit.envolt.mds*), 32
ModeloImpaciente (*class in tinamit.envolt.bf*), 32
ModeloIndeterminado (*class in tinamit.envolt.bf*), 31
ModeloPySD (*class in tinamit.envolt.mds.pysd*), 33
ModeloVensimDLL (*class in tinamit.envolt.mds.vensim_dll*), 33

N

Nivel (*class in tinamit.geog.región*), 39

O

obt_conf() (*tinamit.mod.modelo.Modelo class method*), 37
obt_val() (*tinamit.mod.var.Variable method*), 48
obt_vals() (*tinamit.datos.bd.BD method*), 44
olvidar_envolt_mds() (*in module tinamit.envolt.mds._auto*), 32
OtraForma (*class in tinamit.geog.mapa*), 42

P

paralelizable() (*tinamit.conect.SuperConectado method*), 35
paralelizable() (*tinamit.envolt.mds.pysd.ModeloPySD method*), 33
paralelizable() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 34
paralelizable() (*tinamit.mod.modelo.Modelo method*), 37

pariente() (*tinamit.geog.región.Lugar method*), 39
poner_val() (*tinamit.mod.var.Variable method*), 48
poner_vals_paso() (*tinamit.envolt.bf.VarPasoIndeter method*), 49
prb_egreso() (*tinamit.envolt.bf.ModeloBF class method*), 30
prb_ingreso() (*tinamit.envolt.bf.ModeloBF class method*), 30
prb_simul() (*tinamit.envolt.bf.ModeloBF class method*), 31

R

registrar_envolt_mds() (*in module tinamit.envolt.mds._auto*), 32
reinic() (*tinamit.mod.var.Variable method*), 48
ResultadosGrupo (*class in tinamit.mod.res*), 49
ResultadosSimul (*class in tinamit.mod.res*), 49
ResultadosVar (*class in tinamit.mod.res*), 49

S

SimuladorGeog (*class in tinamit.calibs.geog_mod*), 46
simular() (*tinamit.calibs.geog_mod.SimuladorGeog method*), 46
simular() (*tinamit.mod.modelo.Modelo method*), 37
simular_grupo() (*tinamit.mod.modelo.Modelo method*), 38
SuperConectado (*class in tinamit.conect*), 35

T

tinamit.datos.bd (*module*), 44
tinamit.datos.fuente (*module*), 43
tinamit.envolt.mds._auto (*module*), 32
tinamit.geog.mapa (*module*), 40
tinamit.geog.región (*module*), 38
tinamit.mod.prbs (*module*), 38
tinamit.mod.res (*module*), 49
tinamit.unids.conv (*module*), 50
tinamit.unids.trads (*module*), 50
trad_unid() (*in module tinamit.unids.trads*), 51

U

unidad_tiempo() (*tinamit.conect.SuperConectado method*), 35
unidad_tiempo() (*tinamit.envolt.bf.ModeloBF method*), 31
unidad_tiempo() (*tinamit.envolt.bf.ModeloBloques method*), 31
unidad_tiempo() (*tinamit.envolt.bf.ModeloDeterminado method*), 31
unidad_tiempo() (*tinamit.envolt.bf.ModeloImpaciente method*), 32
unidad_tiempo() (*tinamit.envolt.bf.ModeloIndeterminado method*), 32
unidad_tiempo() (*tinamit.envolt.mds.ModeloDS method*), 32
unidad_tiempo() (*tinamit.envolt.mds.pysd.ModeloPySD method*), 33
unidad_tiempo() (*tinamit.envolt.mds.vensim_dll.ModeloVensimDLL method*), 34
unidad_tiempo() (*tinamit.mod.modelo.Modelo method*), 38

V

ValidadorGeog (*class in tinamit.calibs.geog_mod*), 47
ValidadorMod (*class in tinamit.calibs.valid*), 47
validar() (*tinamit.calibs.geog_mod.ValidadorGeog method*), 47
validar() (*tinamit.calibs.valid.ValidadorMod method*), 47
VarAuxiliar (*class in tinamit.envolt.mds*), 34
VarBloque (*class in tinamit.envolt.bf*), 49
VarConstante (*class in tinamit.envolt.mds*), 34
Variable (*class in tinamit.mod.var*), 48
VariablesMDS (*class in tinamit.envolt.mds*), 34
VariablesMod (*class in tinamit.mod.vars_mod*), 48
VariablesModBloques (*class in tinamit.envolt.bf*), 49
VariablesModDeter (*class in tinamit.envolt.bf*), 48
VariablesModImpaciente (*class in tinamit.envolt.bf*), 48
VariablesModIndeterminado (*class in tinamit.envolt.bf*), 49
VarInic (*class in tinamit.envolt.mds*), 34
VarMDS (*class in tinamit.envolt.mds*), 34
VarNivel (*class in tinamit.envolt.mds*), 34
VarPasoDeter (*class in tinamit.envolt.bf*), 48
VarPasoIndeter (*class in tinamit.envolt.bf*), 49
vars_paso() (*tinamit.envolt.bf.VariablesModImpaciente method*), 48
verificar_leer_egr() (*in module tinamit.mod.prbs*), 38
verificar_leer_ingr() (*in module tinamit.mod.prbs*), 38
verificar_simul() (*in module tinamit.mod.prbs*), 38