Minimal Code Example: Symbolic YIBs Model

Python/Jupyter Setup (No edits)

Jupyter Settings:

In [ ]:
```
#load HTML to adjust jupyter settings
from IPython.display import HTML

#adjust jupyter display to full screen width
display(HTML("<style>.container { width:100% !important; }</style>"))

#set auto reload for notebook
%load_ext autoreload
%autoreload 2
```

Python Packages:

In [1]:
```
# Packages for symbolic code: 
from sympy import Symbol, Function, diag, ImmutableMatrix 
from ComputabilityGraphs.CMTVS import CMTVS
from bgc_md2.helper import module_computers
from bgc_md2.resolve.mvars import (
    InFluxesBySymbol,
    OutFluxesBySymbol,
    InternalFluxesBySymbol,
    TimeSymbol,
    StateVariableTuple
)
from CompartmentalSystems.TimeStepIterator import TimeStepIterator2
import CompartmentalSystems.helpers_reservoir as hr
import bgc_md2.resolve.computers as bgc_c
import bgc_md2.display_helpers as dh
import bgc_md2.helper as h
from collections import namedtuple

# Other packages
import sys
sys.path.insert(0,'..') # necessary to import general_helpers
from general_helpers import (
    download_TRENDY_output,
    day_2_month_index,
    month_2_day_index,
    make_B_u_funcs_2,
    monthly_to_yearly,
    plot_solutions
)
from pathlib import Path
from copy import copy, deepcopy
from functools import reduce
from typing import Callable
from pprint import pprint
import json 
import netCDF4 as nc
import numpy as np
import matplotlib.pyplot as plt
```

Symbolic Setup (Must Edit)

Define Symbols using named tuples for allocation, pools, fluxes, constants:

In [2]:
```
#Create namedtuple of allocation coefficients
Allocation = namedtuple(
    "Allocation",
    [
        'beta_leaf',                  #Names: beta_poolname
        'beta_root',
        'beta_wood'
    ]
)

#Create namedtuple of pools
Pools = namedtuple(
    "Pools",
    [
        'c_leaf',                     #Names: c_poolname
        'c_root',
        'c_wood',
        'c_lit_cwd',
        'c_lit_met',
        'c_lit_str',
        'c_lit_mic',
        'c_soil_met',
        'c_soil_str',
        'c_soil_mic',
        'c_soil_slow',
        'c_soil_passive'
    ]
)

#Create namedtuple of initial pools
InitialPools = namedtuple(
    "InitialPools",
    [
        'c_leaf_0',                   #Names: c_poolname_0
        'c_root_0',
        'c_wood_0',
        'c_lit_cwd_0',
        'c_lit_met_0',
        'c_lit_str_0',
        'c_lit_mic_0',
        'c_soil_met_0',
        'c_soil_str_0',
        'c_soil_mic_0',
        'c_soil_slow_0',
        'c_soil_passive_0'
    ]
)

#Create namedtuple of flux rates leaving the system
FluxRates = namedtuple(
    "FluxRates",
    [
        'r_c_lit_cwd_rh',             #Pools with loss from system will be listed here
        'r_c_lit_met_rh',             #Names: r_c_poolname_rh
        'r_c_lit_str_rh',
        'r_c_lit_mic_rh',
        'r_c_soil_met_rh',
        'r_c_soil_str_rh',
        'r_c_soil_mic_rh',
        'r_c_soil_slow_rh',
        'r_c_soil_passive_rh',
        'r_c_leaf_2_c_lit_met',       #Pool transfer paths
        'r_c_leaf_2_c_lit_str',       #Names: r_c_donorPool_2_recievingPool
```

```
            'r_c_root_2_c_soil_met',
            'r_c_root_2_c_soil_str',
            'r_c_wood_2_c_lit_cwd',
            'r_c_lit_cwd_2_c_lit_mic',
            'r_c_lit_cwd_2_c_soil_slow',
            'r_c_lit_met_2_c_lit_mic',
            'r_c_lit_str_2_c_lit_mic',
            'r_c_lit_str_2_c_soil_slow',
            'r_c_lit_mic_2_c_soil_slow',
            'r_c_soil_met_2_c_soil_mic',
            'r_c_soil_str_2_c_soil_mic',
            'r_c_soil_str_2_c_soil_slow',
            'r_c_soil_mic_2_c_soil_slow',
            'r_c_soil_mic_2_c_soil_passive',
            'r_c_soil_slow_2_c_soil_mic',
            'r_c_soil_slow_2_c_soil_passive',
            'r_c_soil_passive_2_c_soil_mic'
        ]
)

#define time
Time = namedtuple(
    "Time",
    ['t']
)

#Create namedtuple of constants used in model
Constants = namedtuple(
    "Constants",
    [
        'npp_0',        #Initial input/pools
        'rh_0',
        'c_veg_0',
        'c_soil_0',
        'clay',         #Constants like clay
        'silt',
        'nyears'        #Run time (years for my model)
    ]
)

#Combine all named tuples to create symbols
Symbols = namedtuple(
    "Symbols",
    Allocation._fields + Pools._fields + InitialPools._fields + \
    FluxRates._fields + Time._fields + Constants._fields
)

#Create symbols
for k in Symbols._fields:
    code=k+" = Symbol('{0}')".format(k)
    exec(code)

#define beta wood from other allocation values
beta_wood = 1.0-(beta_leaf+beta_root)
```

Create functions for environmental scaler and input:

In [3]:
```
#create symbols for scaler and input functions
func_dict={
    'xi': 'Environmental scaler as a function of time',
    'NPP': 'Inputs as a function of time',
}
for k in func_dict.keys():
    code=k+" = Function('{0}')".format(k)
    exec(code)
```

```
#define t as a symbol for time
t=TimeSymbol("t")
```

Symbolic Model Description (Must Edit)

Define your model using sympy:

In [4]:
```
#define model in sympy format
mvs = CMTVS(
    {
        t,
        StateVariableTuple(
            #Define State variables
            Pools._fields
        ),
        InFluxesBySymbol(
            {    #define input/allocation
                #RecievingPool: Input * Allocation
                c_leaf: NPP(t) * beta_leaf,
                c_root: NPP(t) * beta_root,
                c_wood: NPP(t) * beta_wood
            }
        ),
        OutFluxesBySymbol(
            {    #define fluxes leaving the system
                #Fluxes leaving the system: FluRate * DonorPool * EnvironmentalScaler
                c_lit_cwd: r_c_lit_cwd_rh * c_lit_cwd * xi(t),
                c_lit_met: r_c_lit_met_rh * c_lit_met * xi(t),
                c_lit_str: r_c_lit_str_rh * c_lit_str * xi(t),
                c_lit_mic: r_c_lit_mic_rh * c_lit_mic * xi(t),
                c_soil_met: r_c_soil_met_rh * c_soil_met * xi(t),
                c_soil_str: r_c_soil_str_rh * c_soil_str * xi(t),
                c_soil_mic: r_c_soil_mic_rh * c_soil_mic * xi(t),
                c_soil_slow: r_c_soil_slow_rh * c_soil_slow * xi(t),
                c_soil_passive: r_c_soil_passive_rh * c_soil_passive * xi(t),
            }
        ),
        InternalFluxesBySymbol(
            {    #define fluxes between pools
                #(Donor pool, recieving pool): FluxRate * DonorPool
                (c_leaf, c_lit_met): r_c_leaf_2_c_lit_met * c_leaf,
                (c_leaf, c_lit_str): r_c_leaf_2_c_lit_str * c_leaf,
                (c_root, c_soil_met): r_c_root_2_c_soil_met * c_root,
                (c_root, c_soil_str): r_c_root_2_c_soil_str * c_root,
                (c_wood, c_lit_cwd): r_c_wood_2_c_lit_cwd * c_wood,
                (c_lit_cwd, c_lit_mic): r_c_lit_cwd_2_c_lit_mic * c_lit_cwd * xi(t),
                (c_lit_cwd, c_soil_slow): r_c_lit_cwd_2_c_soil_slow * c_lit_cwd * xi(t),
                (c_lit_met, c_lit_mic): r_c_lit_met_2_c_lit_mic * c_lit_met * xi(t),
                (c_lit_str, c_lit_mic): r_c_lit_str_2_c_lit_mic * c_lit_str * xi(t),
                (c_lit_str, c_soil_slow): r_c_lit_str_2_c_soil_slow * c_lit_str * xi(t),
                (c_lit_mic, c_soil_slow): r_c_lit_mic_2_c_soil_slow * c_lit_mic * xi(t),
                (c_soil_met, c_soil_mic): r_c_soil_met_2_c_soil_mic * c_soil_met * xi(t)
                (c_soil_str, c_soil_mic): r_c_soil_str_2_c_soil_mic * c_soil_str * xi(t)
                (c_soil_str, c_soil_slow): r_c_soil_str_2_c_soil_slow * c_soil_str* xi(t
                (c_soil_mic, c_soil_slow): r_c_soil_mic_2_c_soil_slow * c_soil_mic* xi(t
                (c_soil_mic, c_soil_passive): r_c_soil_mic_2_c_soil_passive * c_soil_mic
                (c_soil_slow, c_soil_mic): r_c_soil_slow_2_c_soil_mic * c_soil_slow * xi
                (c_soil_slow, c_soil_passive): r_c_soil_slow_2_c_soil_passive * c_soil_s
                (c_soil_passive, c_soil_mic): r_c_soil_passive_2_c_soil_mic * c_soil_pas
            }
        ),
    },
```

```
    computers=module_computers(bgc_c)
)
```

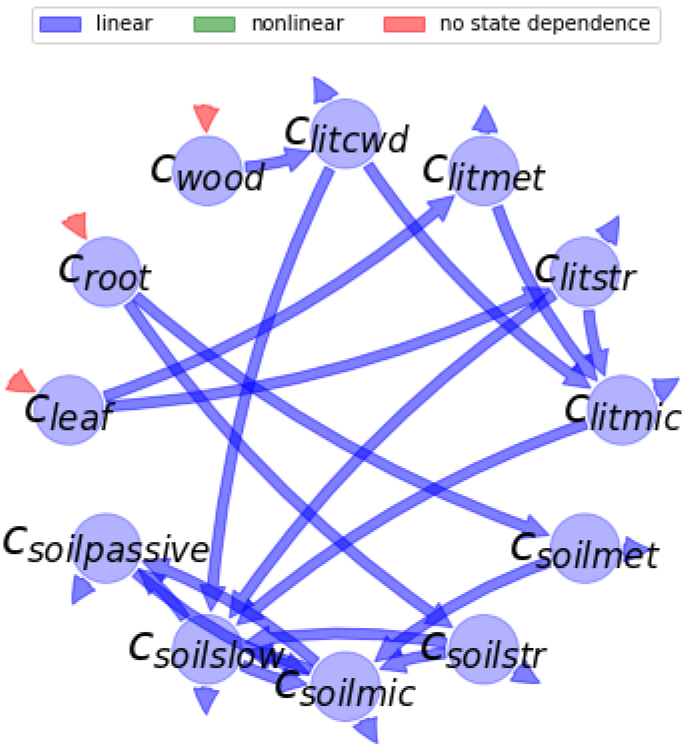
Model Figure and Matrix Equations

Model Figure:

In [5]:
```
h.compartmental_graph(mvs)
```

```
{CompartmentalMatrix,InputTuple}
{InternalFluxesBySymbol,OutFluxesBySymbol,InputTuple,SmoothReservoirModel,InFluxesBySymb
ol,CompartmentalMatrix}
{InternalFluxesBySymbol,OutFluxesBySymbol,InputTuple,SmoothReservoirModel,InFluxesBySymb
ol,CompartmentalMatrix}
```

Out[5]:



Matrix equations:

In [6]:
```
dh.mass_balance_equation(mvs)
```

Out[6]:

$$
\frac{dx}{dt} = \begin{bmatrix} \beta_{leaf}\,\mathrm{NPP}\,(t) \\ \beta_{root}\,\mathrm{NPP}\,(t) \\ \left(-\beta_{leaf} - \beta_{root} + 1.0\right)\,\mathrm{NPP}\,(t) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}
$$

$$
+
$$

$$
\begin{bmatrix} -r_{cleaf2clitmet} - r_{cleaf2clitstr} & 0 & 0 & 0 \\ 0 & -r_{croot2csoilmet} - r_{croot2csoilstr} & 0 & 0 \\ 0 & 0 & -r_{cwood2clitcwd} & 0 \\ 0 & 0 & r_{cwood2clitcwd} & -(r_{clitcwd2clitmic} + r_{clitcwd2c} \\ r_{cleaf2clitmet} & 0 & 0 & 0 \\ r_{cleaf2clitstr} & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{clitcwd2clit} \\ 0 & r_{croot2csoilmet} & 0 & 0 \\ 0 & r_{croot2csoilstr} & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{clitcwd2csoil} \\ 0 & 0 & 0 & 0 \end{matrix}
$$

Download Data (Must Edit)

TRENDY Data

Make sure you have a config.json file in your model folder:
Config.jon file contents: `{"username": "trendy-v9", "password": "gcb-2020", "dataPath": "/path/to/data/folder"}`

In [7]:
```
# Read username, password, dataPath from config.json file
with Path('config.json').open(mode='r') as f:
    conf_dict=json.load(f)

# Create tuples of data streams
# For YIBs I have annual and monthly file names so I defined them separately
# If all your data is similarly named this would be a single "observables" tuple

# Annual data streams on TRENDY server
observables_annual = namedtuple(
    'observables_annual',
    ["cVeg", "cSoil"]
)
# Monthly data streams on TRENDY server
observables_monthly = namedtuple(
    'observables_monthly',
    ["rh"]
```

```
)
# Combine tuples to request data from TRENDY server
observables = namedtuple(
    "observables",
    observables_annual._fields+observables_monthly._fields
)
# Driver data streams on TRENDY server
drivers=namedtuple(
    "drivers",
    ["npp"]
)

#when downloading data make sure model names match TRENDY server names:
#"CABLE-POP","CLASSIC","CLM5","DLEM","IBIS","ISAM","ISBA_CTRIP",
#"JSBACH","JULES-ES-1.0","LPJ-GUESS","LPJwsl","LPX-Bern","OCN",
#"ORCHIDEE","ORCHIDEE-CNP","ORCHIDEEv3","ORCHIDEEv3_0.5deg"
#"SDGVM","VISIT","YIBs"

# Define function to download data from TRENDY server
def download_my_TRENDY_output():
    download_TRENDY_output(
        username=conf_dict["username"],
        password=conf_dict["password"],
        dataPath=Path(conf_dict["dataPath"]),#platform independent path desc. (Windows v
        models=['YIBs'],
        variables =observables._fields + drivers._fields
    )

# call above function to download TRENDY data
# This can takes a minute to connect to the server
# The download itself can take hours depending on internet speed.
# If "can't connect to TRENDY server" error occurs, try again later.
download_my_TRENDY_output()
```

```
downloading data for YIBs model
C:\bgc_md2\prototypes\working_group_2021\jon_yib\TRENDY_data\YIBs_S2_Annual_cVeg.nc exis
ts, skipping
C:\bgc_md2\prototypes\working_group_2021\jon_yib\TRENDY_data\YIBs_S2_Annual_cSoil.nc exi
sts, skipping
C:\bgc_md2\prototypes\working_group_2021\jon_yib\TRENDY_data\YIBs_S2_Monthly_rh.nc exist
s, skipping
C:\bgc_md2\prototypes\working_group_2021\jon_yib\TRENDY_data\YIBs_S2_Monthly_npp.nc exis
ts, skipping
finished!
```

Connect Data and Symbols (Must Edit)

Define function to subset netCDF files and link to data symbols:

In [8]:

```
# Define function to download, scale, map TRENDY data
def get_example_site_vars(dataPath):

    # Define single geospatial cell
    s = slice(None, None, None)  # this is the same as :
    t = s, 74, 118  # [t] = [:,49,325]

    # Define function to select geospatial cell and scale data
    def f(tup):
        vn, fn = tup
        path = dataPath.joinpath(fn)
        # Read NetCDF data but only at the point where we want them
        ds = nc.Dataset(str(path))
        #check for npp/gpp/rh/ra to convert from kg/m2/s to kg/m2/day
        if vn in ["npp","gpp","rh","ra"]:
```

```
            return ds.variables[vn][t]*86400
        else:
            return ds.variables[vn][t]

    # Link symbols and data:
    # YIBS has annual vs monthly file names so they are linked separately
    # If all your data is similarly named you can do this in one step

    # Create annual file names (single step if files similarly named)
    o_names=[(f,"YIBs_S2_Annual_{}.nc".format(f)) for f in observables_annual._fields]

    # Create monthly file names (can remove if done in one step above)
    monthly_names=[(f,"YIBs_S2_Monthly_{}.nc".format(f)) for f in observables_monthly._f
    # Extend name list with monthly names
    o_names.extend(monthly_names)

    # create file names for drivers
    d_names=[(f,"YIBs_S2_Monthly_{}.nc".format(f)) for f in drivers._fields]

    # Link symbols and data for observables/drivers
    return (observables(*map(f, o_names)),drivers(*map(f,d_names)))

#call function to link symbols and data
svs,dvs=get_example_site_vars(dataPath=Path(conf_dict["dataPath"]))

#look at data
svs,dvs
```

Out[8]:

```
(observables(cVeg=masked_array(data=[0.79001224, 0.78839403, 0.78843683, 0.7879241 ,
                    0.7894626 , 0.78924614, 0.7896311 , 0.7902869 ,
                    0.7912474 , 0.7897687 , 0.7908123 , 0.79213333,
                    0.79134333, 0.79214597, 0.7905707 , 0.78847754,
                    0.78959256, 0.79222375, 0.7900269 , 0.78920066,
                    0.79027295, 0.78867215, 0.7887283 , 0.78822595,
                    0.7897618 , 0.78956115, 0.7899609 , 0.79061717,
                    0.7915832 , 0.7901114 , 0.7911532 , 0.79245937,
                    0.79166096, 0.7924398 , 0.7908483 , 0.7887362 ,
                    0.78982097, 0.7924049 , 0.79016876, 0.7893033 ,
                    0.79032636, 0.7886797 , 0.78868663, 0.78813314,
                    0.78962505, 0.7893757 , 0.7897286 , 0.7903508 ,
                    0.7912823 , 0.78977495, 0.7907901 , 0.792084  ,
                    0.791266  , 0.7920469 , 0.7904454 , 0.78832304,
                    0.7894133 , 0.79203063, 0.7898087 , 0.78894806,
                    0.7900081 , 0.78838766, 0.7884254 , 0.78791547,
                    0.7894603 , 0.7892559 , 0.78965807, 0.7903336 ,
                    0.7913177 , 0.7898686 , 0.79094166, 0.7922875 ,
                    0.7915299 , 0.7923593 , 0.7908216 , 0.7887676 ,
                    0.7899211 , 0.7925701 , 0.79041404, 0.7896521 ,
                    0.7907582 , 0.7892155 , 0.78933954, 0.78889203,
                    0.7904626 , 0.7903328 , 0.7908125 , 0.791524  ,
                    0.7925716 , 0.79117936, 0.79230833, 0.79365873,
                    0.79295266, 0.793788  , 0.7923034 , 0.79034543,
                    0.7915509 , 0.79414254, 0.7920412 , 0.7914079 ,
                    0.79248947, 0.7909959 , 0.7911535 , 0.7906851 ,
                    0.79216015, 0.792016  , 0.7924777 , 0.79310876,
                    0.79409903, 0.7926451 , 0.7937013 , 0.79492795,
                    0.7941424 , 0.7948638 , 0.79330593, 0.7912884 ,
                    0.7924068 , 0.794862  , 0.7926755 , 0.7919831 ,
                    0.79296154, 0.7913875 , 0.7914697 , 0.7909215 ,
                    0.7923209 , 0.7921204 , 0.79253614, 0.7931287 ,
                    0.7940936 , 0.792623  , 0.79367256, 0.79490256,
                    0.7941256 , 0.79486513, 0.793334  , 0.7913543 ,
                    0.7925194 , 0.7950236 , 0.79290724, 0.7923094 ,
                    0.79336596, 0.791904  , 0.7921035 , 0.79165846,
                    0.7931265 , 0.79302645, 0.7935335 , 0.79416496,
                    0.79517746, 0.79374254, 0.79480267, 0.7959927 ,
```

```
                                0.7952057 , 0.7958896 , 0.79433286, 0.7923396 ,
                                0.7934661 , 0.7958686 , 0.7937089 , 0.79309475,
                                0.79407376, 0.792558  , 0.7927037 , 0.7921926 ,
                                0.7935918 , 0.7934637 , 0.793959  , 0.7945831 ,
                                0.7956158 , 0.7942158 , 0.79532963, 0.7965671 ,
                                0.7958591 , 0.7966048 , 0.7951404 , 0.7932631 ,
                                0.7944802 , 0.7969076 , 0.7948477 , 0.7943862 ,
                                0.7954159 , 0.79401994, 0.7942788 , 0.79384106,
                                0.7952549 , 0.7952248 , 0.7958209 , 0.7964784 ,
                                0.7975814 , 0.7962497 , 0.7974246 , 0.79864335,
                                0.7979903 , 0.7987256 , 0.7973333 , 0.7955747 ,
                                0.7968481 , 0.79917276, 0.7971717 , 0.7968565 ,
                                0.7978275 , 0.7964826 , 0.7967817 , 0.7963165 ,
                                0.79761237, 0.7975847 , 0.79817873, 0.7987531 ,
                                0.7998231 , 0.7984621 , 0.79960114, 0.8007167 ,
                                0.8000478 , 0.80071855, 0.7993476 , 0.7976669 ,
                                0.7989807 , 0.80122626, 0.79930586, 0.7991597 ,
                                0.80011714, 0.7992416 , 0.79921895, 0.79939204,
                                0.80072963, 0.8012495 , 0.7998112 , 0.79929745,
                                0.80000156, 0.8006911 , 0.8005191 , 0.8004204 ,
                                0.80023384, 0.80087423, 0.80214   , 0.80194074,
                                0.8029987 , 0.8013706 , 0.8001357 , 0.8009014 ,
                                0.8015491 , 0.800601  , 0.80154353, 0.8015747 ,
                                0.80161166, 0.8009643 , 0.80212384, 0.8035053 ,
                                0.80252725, 0.8027026 , 0.8022803 , 0.8022153 ,
                                0.80383855, 0.80215204, 0.80273193, 0.80256563,
                                0.8031648 , 0.8026056 , 0.8036707 , 0.8041024 ,
                                0.80397505, 0.80378014, 0.8030848 , 0.80236906,
                                0.8026123 , 0.80277854, 0.8028762 , 0.80307   ,
                                0.8052032 , 0.80381715, 0.8033888 , 0.8052897 ,
                                0.8062391 , 0.8075465 , 0.809064  , 0.8086634 ,
                                0.80810124, 0.8092197 , 0.81094265, 0.8111933 ,
                                0.811473  , 0.81206256, 0.8121969 , 0.8107974 ,
                                0.8114987 , 0.8121211 , 0.8131765 , 0.81131077,
                                0.81155336, 0.81224   , 0.81309223, 0.8132762 ,
                                0.81404305, 0.81318086, 0.8132129 , 0.8152458 ,
                                0.81519765, 0.81541026, 0.8148192 , 0.8156285 ,
                                0.81625366, 0.81740195, 0.8176236 , 0.8177754 ,
                                0.81835526, 0.81910896, 0.81962126, 0.81922305,
                                0.81971765, 0.8201435 , 0.8206032 , 0.82023627,
                                0.82122886, 0.82194775, 0.8212936 , 0.8220845 ,
                                0.82193154, 0.8223047 , 0.8222557 , 0.8229839 ],
                         mask=False,
                   fill_value=1e+20,
                        dtype=float32), cSoil=masked_array(data=[15.38775  , 15.390659 , 15.370254
5, 15.339091 ,
                                15.36509  , 15.372412 , 15.368353 , 15.384551 ,
                                15.394104 , 15.372406 , 15.38862  , 15.432338 ,
                                15.461357 , 15.499283 , 15.52114  , 15.512947 ,
                                15.512942 , 15.545851 , 15.545021 , 15.509817 ,
                                15.497677 , 15.498684 , 15.476567 , 15.443854 ,
                                15.468583 , 15.474821 , 15.469921 , 15.485404 ,
                                15.494335 , 15.471999 , 15.487732 , 15.5309725,
                                15.559393 , 15.596488 , 15.617353 , 15.607888 ,
                                15.606373 , 15.637452 , 15.634401 , 15.596653 ,
                                15.581643 , 15.579563 , 15.554075 , 15.5176935,
                                15.538567 , 15.54074  , 15.531581 , 15.542726 ,
                                15.547336 , 15.520697 , 15.532087 , 15.571075 ,
                                15.595395 , 15.628746 , 15.64611  , 15.633494 ,
                                15.629133 , 15.657746 , 15.652734 , 15.613371 ,
                                15.597192 , 15.594312 , 15.568476 , 15.532274 ,
                                15.553756 , 15.556981 , 15.549319 , 15.562407 ,
                                15.569359 , 15.545497 , 15.560091 , 15.602641 ,
                                15.6309   , 15.668426 , 15.690281 , 15.682373 ,
                                15.683033 , 15.716915 , 15.717252 , 15.683651 ,
                                15.673455 , 15.6768675, 15.657548 , 15.627902 ,
```

```
                    15.656173 , 15.66636  , 15.665969 , 15.68642  ,
                    15.700794 , 15.684052 , 15.706266 , 15.756512 ,
                    15.79229  , 15.836843 , 15.865675 , 15.864575 ,
                    15.872183 , 15.912693 , 15.919092 , 15.891912 ,
                    15.887856 , 15.897375 , 15.883669 , 15.858652 ,
                    15.89112  , 15.904748 , 15.907368 , 15.930071 ,
                    15.945852 , 15.929152 , 15.951188 , 16.0005   ,
                    16.034342 , 16.075806 , 16.100878 , 16.095192 ,
                    16.09768  , 16.132273 , 16.131947 , 16.09766  ,
                    16.086033 , 16.087831 , 16.066082 , 16.032688 ,
                    16.056854 , 16.062153 , 16.056578 , 16.07136  ,
                    16.079617 , 16.055737 , 16.071177 , 16.114502 ,
                    16.142921 , 16.17962  , 16.20064  , 16.191639 ,
                    16.19162  , 16.22454  , 16.223402 , 16.189339 ,
                    16.178917 , 16.183065 , 16.164705 , 16.135473 ,
                    16.164618 , 16.1755   , 16.176102 , 16.197195 ,
                    16.21169  , 16.19357  , 16.214777 , 16.263382 ,
                    16.29635  , 16.336548 , 16.360348 , 16.353245 ,
                    16.354584 , 16.388054 , 16.386524 , 16.351639 ,
                    16.339773 , 16.34202  , 16.321157 , 16.288769 ,
                    16.3146   , 16.322079 , 16.31945  , 16.337671 ,
                    16.349834 , 16.329845 , 16.35024  , 16.399014 ,
                    16.433048 , 16.475023 , 16.501505 , 16.497864 ,
                    16.503492 , 16.541737 , 16.545296 , 16.516445 ,
                    16.511106 , 16.520685 , 16.507597 , 16.48292  ,
                    16.516947 , 16.532888 , 16.539373 , 16.566872 ,
                    16.588373 , 16.577148 , 16.607195 , 16.665691 ,
                    16.709099 , 16.759628 , 16.794706 , 16.799397 ,
                    16.813614 , 16.859818 , 16.870346 , 16.84889  ,
                    16.850405 , 16.86689  , 16.860163 , 16.840591 ,
                    16.879375 , 16.899353 , 16.909725 , 16.940414 ,
                    16.964449 , 16.954376 , 16.986105 , 17.046043 ,
                    17.090303 , 17.140842 , 17.176151 , 17.181116 ,
                    17.19643  , 17.243881 , 17.255621 , 17.236744 ,
                    17.241247 , 17.264427 , 17.3059   , 17.304445 ,
                    17.298433 , 17.315159 , 17.335995 , 17.309258 ,
                    17.315746 , 17.335497 , 17.365124 , 17.367481 ,
                    17.38305  , 17.383047 , 17.412819 , 17.442453 ,
                    17.47014  , 17.49998  , 17.526888 , 17.546316 ,
                    17.567009 , 17.574799 , 17.61128  , 17.6421   ,
                    17.682055 , 17.706642 , 17.736448 , 17.826115 ,
                    17.865616 , 17.903963 , 17.92235  , 17.93592  ,
                    17.988188 , 18.021055 , 18.03428  , 18.053186 ,
                    18.073069 , 18.102219 , 18.12996  , 18.192606 ,
                    18.235762 , 18.267065 , 18.281713 , 18.283396 ,
                    18.261335 , 18.275835 , 18.289845 , 18.293907 ,
                    18.337074 , 18.352913 , 18.338028 , 18.351826 ,
                    18.406359 , 18.4699   , 18.54703  , 18.626944 ,
                    18.716755 , 18.792309 , 18.887589 , 18.99275  ,
                    19.05989  , 19.14294  , 19.241932 , 19.312763 ,
                    19.390385 , 19.48051  , 19.560171 , 19.622593 ,
                    19.68967  , 19.750666 , 19.841103 , 19.916422 ,
                    19.979996 , 20.062853 , 20.131815 , 20.175283 ,
                    20.246557 , 20.331661 , 20.410244 , 20.486975 ,
                    20.58424  , 20.652264 , 20.75402  , 20.852797 ,
                    20.906578 , 20.961012 , 21.05373  , 21.149122 ,
                    21.210941 , 21.291107 , 21.381247 , 21.484568 ,
                    21.591774 , 21.71492  , 21.84473  , 21.967419 ,
                    22.05246  , 22.154308 , 22.26017  , 22.359509 ],
              mask=False,
        fill_value=1e+20,
             dtype=float32), rh=masked_array(data=[0.00517356, 0.00521828, 0.00526037,
..., 0.00629746,
                    0.00695491, 0.00745832],
              mask=False,
        fill_value=1e+20)),
```

```
drivers(npp=masked_array(data=[0.00538536, 0.00513949, 0.00475123, ..., 0.00721739,
                               0.00793872, 0.00799198],
                         mask=False,
                   fill_value=1e+20)))
```

Create Symbols for ξ, K, and A (No Edits)

Setup Yiqi matrix format:

In [9]:
```
sv=mvs.get_StateVariableTuple()                              # Get state variables from sy
n=len(sv)                                                    # Count number of pools
srm = mvs.get_SmoothReservoirModel()                         # Create smooth resevoir mode
_,A,N,_,_=srm.xi_T_N_u_representation(factor_out_xi=False) # Create A and N symbols
```

ξ Matrix:

In [10]:
```
# Create environmental scaler matrix
xi_d=diag([1,1,1]+[xi(t) for i in range(n-3)],unpack=True)
xi_d
```

Out[10]:

$$\left[\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & \xi(t) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & \xi(t) & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & \xi(t) & 0 & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & \xi(t) & 0 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \xi(t) & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \xi(t) & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \xi(t) & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \xi(t) & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \xi(t)
\end{array}\right]$$

K Matrix:

In [11]:
```
# Create empty K matrix
K=xi_d.inv()*N
# Create new symbols for the k_{i}
for i in range(n):
    if K[i,i]!=0:
        name="k_{0}".format(sv[i])
        code="{0}=Symbol('{0}')".format(name)
        #print(code)
        exec(code)

# Create $K$ matrix
K_sym=ImmutableMatrix(
    n,n,
    lambda i,j: Symbol("k_" + str(sv[i])) if i==j else 0
)
K_sym
```

Out[11]:

$$\begin{bmatrix} k_{cleaf} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{croot} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{cwood} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & k_{clitcwd} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & k_{clitmet} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & k_{clitstr} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & k_{clitmic} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{csoilmet} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{csoilstr} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{csoilmic} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{csoilslow} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & k_{cs} \end{bmatrix}$$

f symbols in A Matrix:

In [12]:

```
# Create new symbols for the f_{i,j}
for i in range(n):
    for j in range(n):
        if A[i,j]!=0 and i!=j:
            name="f_" + str(sv[j]) + "_2_" + str(sv[i])
            code="{0}=Symbol('{0}')".format(name)
            #print(code)
            exec(code)

# Place $f$ values in $A$ matrix
A_sym=ImmutableMatrix(
    n,n,
    lambda i,j:  -1 if i==j else (
        0 if A[i,j]==0 else Symbol("f_" + str(sv[j]) + "_2_" + str(sv[i]))
    )
)
A_sym
```

Out[12]:

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & f_{cwood2clitcwd} & -1 & 0 & 0 & 0 \\ f_{cleaf2clitmet} & 0 & 0 & 0 & -1 & 0 & 0 \\ f_{cleaf2clitstr} & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & f_{clitcwd2clitmic} & f_{clitmet2clitmic} & f_{clitstr2clitmic} & -1 \\ 0 & f_{croot2csoilmet} & 0 & 0 & 0 & 0 & 0 \\ 0 & f_{croot2csoilstr} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{clitcwd2csoilslow} & 0 & f_{clitstr2csoilslow} & f_{clitmic2csoils} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A matrix:

In [13]:

```
# Create A matrix
M_sym=A_sym*K_sym
M_sym
```

Out[13]:

$$\begin{bmatrix} -k_{cleaf} & 0 & 0 & 0 & 0 \\ 0 & -k_{croot} & 0 & 0 & 0 \\ 0 & 0 & -k_{cwood} & 0 & 0 \\ 0 & 0 & f_{cwood2clitcwd}k_{cwood} & -k_{clitcwd} & 0 \\ f_{cleaf2clitmet}k_{cleaf} & 0 & 0 & 0 & -k_{clitmet} \\ f_{cleaf2clitstr}k_{cleaf} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{clitcwd2clitmic}k_{clitcwd} & f_{clitmet2clitmic}k_{clitmet} \\ 0 & f_{croot2csoilmet}k_{croot} & 0 & 0 & 0 \\ 0 & f_{croot2csoilstr}k_{croot} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & f_{clitcwd2csoilslow}k_{clitcwd} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Create Dictionary of All Fluxes (No Edits)

In [14]:

```
# Create a dictionary for the external and internal fluxes (flux divided by dono pool co
outflux_rates = {"r_"+str(key)+"_rh":value/key for key,value in hr.out_fluxes_by_symbol(
internal_flux_rates = {"r_"+str(key[0])+"_2_"+str(key[1]):value/key[0] for key,value in

# Create dictionary of all flux rates
all_rates=deepcopy(outflux_rates)
all_rates.update(internal_flux_rates)
all_rates
```

Out[14]:

```
{'r_c_leaf_rh': k_c_leaf*(-f_c_leaf_2_c_lit_met - f_c_leaf_2_c_lit_str + 1),
 'r_c_root_rh': k_c_root*(-f_c_root_2_c_soil_met - f_c_root_2_c_soil_str + 1),
 'r_c_wood_rh': k_c_wood*(1 - f_c_wood_2_c_lit_cwd),
 'r_c_lit_cwd_rh': k_c_lit_cwd*(-f_c_lit_cwd_2_c_lit_mic - f_c_lit_cwd_2_c_soil_slow +
1),
 'r_c_lit_met_rh': k_c_lit_met*(1 - f_c_lit_met_2_c_lit_mic),
 'r_c_lit_str_rh': k_c_lit_str*(-f_c_lit_str_2_c_lit_mic - f_c_lit_str_2_c_soil_slow +
1),
 'r_c_lit_mic_rh': k_c_lit_mic*(1 - f_c_lit_mic_2_c_soil_slow),
 'r_c_soil_met_rh': k_c_soil_met*(1 - f_c_soil_met_2_c_soil_mic),
 'r_c_soil_str_rh': k_c_soil_str*(-f_c_soil_str_2_c_soil_mic - f_c_soil_str_2_c_soil_slo
w + 1),
 'r_c_soil_mic_rh': k_c_soil_mic*(-f_c_soil_mic_2_c_soil_passive - f_c_soil_mic_2_c_soil
_slow + 1),
 'r_c_soil_slow_rh': k_c_soil_slow*(-f_c_soil_slow_2_c_soil_mic - f_c_soil_slow_2_c_soil
_passive + 1),
 'r_c_soil_passive_rh': k_c_soil_passive*(1 - f_c_soil_passive_2_c_soil_mic),
 'r_c_leaf_2_c_lit_met': f_c_leaf_2_c_lit_met*k_c_leaf,
 'r_c_leaf_2_c_lit_str': f_c_leaf_2_c_lit_str*k_c_leaf,
 'r_c_root_2_c_soil_met': f_c_root_2_c_soil_met*k_c_root,
 'r_c_root_2_c_soil_str': f_c_root_2_c_soil_str*k_c_root,
 'r_c_wood_2_c_lit_cwd': f_c_wood_2_c_lit_cwd*k_c_wood,
 'r_c_lit_cwd_2_c_lit_mic': f_c_lit_cwd_2_c_lit_mic*k_c_lit_cwd,
 'r_c_lit_cwd_2_c_soil_slow': f_c_lit_cwd_2_c_soil_slow*k_c_lit_cwd,
 'r_c_lit_met_2_c_lit_mic': f_c_lit_met_2_c_lit_mic*k_c_lit_met,
 'r_c_lit_str_2_c_lit_mic': f_c_lit_str_2_c_lit_mic*k_c_lit_str,
 'r_c_lit_str_2_c_soil_slow': f_c_lit_str_2_c_soil_slow*k_c_lit_str,
 'r_c_lit_mic_2_c_soil_slow': f_c_lit_mic_2_c_soil_slow*k_c_lit_mic,
 'r_c_soil_met_2_c_soil_mic': f_c_soil_met_2_c_soil_mic*k_c_soil_met,
 'r_c_soil_str_2_c_soil_mic': f_c_soil_str_2_c_soil_mic*k_c_soil_str,
 'r_c_soil_str_2_c_soil_slow': f_c_soil_str_2_c_soil_slow*k_c_soil_str,
 'r_c_soil_mic_2_c_soil_slow': f_c_soil_mic_2_c_soil_slow*k_c_soil_mic,
 'r_c_soil_mic_2_c_soil_passive': f_c_soil_mic_2_c_soil_passive*k_c_soil_mic,
 'r_c_soil_slow_2_c_soil_mic': f_c_soil_slow_2_c_soil_mic*k_c_soil_slow,
```

```
 'r_c_soil_slow_2_c_soil_passive': f_c_soil_slow_2_c_soil_passive*k_c_soil_slow,
 'r_c_soil_passive_2_c_soil_mic': f_c_soil_passive_2_c_soil_mic*k_c_soil_passive}
```

Calculate Rates from f and k values (Must Edit)

I have k and f values describing my model. we can define them here and use them to assign values to rs

In [15]:
```
ParameterValues = namedtuple(
    "ParameterValues",
    [
        "beta_leaf",
        "beta_root",
        "clay",
        "silt",
        "k_leaf",
        "k_root",
        "k_wood",
        "k_cwd",
        "k_samet",
        "k_sastr",
        "k_samic",
        "k_slmet",
        "k_slstr",
        "k_slmic",
        "k_slow",
        "k_arm",
        "f_samet_leaf",
        "f_slmet_root",
        "f_samic_cwd",
    ]
)

epa0 = ParameterValues(
    beta_leaf=0.21,
    beta_root=0.27,
    clay=0.2028,
    silt=0.2808,
    k_leaf=0.014,
    k_root=0.022,
    k_wood=0.003,
    k_cwd=0.005,
    k_samet=0.01,
    k_sastr=0.001,
    k_samic=0.05,
    k_slmet=0.040,
    k_slstr=0.0039,
    k_slmic=0.005,
    k_slow=0.00001,
    k_arm=3.27E-06,
    f_samet_leaf=0.28,
    f_slmet_root=0.34,
    f_samic_cwd=0.29,
)

old_par_dict = {
    'k_c_leaf': 0.014, # define all k values
    'k_c_wood': 0.003,
    'k_c_root': 0.022,
    'k_c_lit_cwd': 0.005,
    'k_c_lit_met': 0.01,
    'k_c_lit_str': 0.001,
    'k_c_lit_mic': 0.05,
    'k_c_soil_met': 0.040,
```

```python
    'k_c_soil_str': 0.0039,
    'k_c_soil_mic': 0.005,
    'k_c_soil_slow': 0.00001,
    'k_c_soil_passive': 3.27E-06,
    'f_c_leaf_2_c_lit_met': epa0.f_samet_leaf,    #define all f values
    'f_c_root_2_c_soil_met': epa0.f_slmet_root,
    'f_c_lit_cwd_2_c_lit_mic': epa0.f_samic_cwd,
    'f_c_leaf_2_c_lit_str': (1-epa0.f_samet_leaf) * epa0.k_leaf,
    'f_c_root_2_c_soil_str': (1-epa0.f_slmet_root) * epa0.k_root,
    'f_c_wood_2_c_lit_cwd': 0.4 * epa0.k_wood,
    'f_c_lit_cwd_2_c_soil_slow': (1-epa0.f_samic_cwd) * epa0.k_cwd,
    'f_c_lit_met_2_c_lit_mic': 0.3 * epa0.k_samet,
    'f_c_lit_str_2_c_lit_mic': 0.1 * epa0.k_sastr,
    'f_c_lit_str_2_c_soil_slow': 0.1 * epa0.k_sastr,
    'f_c_lit_mic_2_c_soil_slow': 0.1 * epa0.k_samic,
    'f_c_soil_met_2_c_soil_mic': 0.4 * epa0.k_slmet,
    'f_c_soil_str_2_c_soil_mic': 0.3 * epa0.k_slstr,
    'f_c_soil_str_2_c_soil_slow': 0.2 * epa0.k_slstr,
    'f_c_soil_mic_2_c_soil_slow': 0.4 * epa0.k_slmic,
    'f_c_soil_mic_2_c_soil_passive': 0.4 * epa0.k_slmic,
    'f_c_soil_slow_2_c_soil_mic': 0.10 * epa0.k_slow,
    'f_c_soil_slow_2_c_soil_passive': 0.45*(0.003+0.009*epa0.clay) * epa0.k_slow,
    'f_c_soil_passive_2_c_soil_mic': 0.10 * epa0.k_arm,
}

# Define allocation parameters to be optimized
par_dict = {
    'beta_leaf': epa0.beta_leaf,
    'beta_root': epa0.beta_root,
}

# translate rates from previous parameters to create dictionary of rates to optimize
par_dict.update(
    {str(k):v.subs(old_par_dict) for k,v in all_rates.items()}
)

# Create namedtuple of parameters to optimize and their translated values
makeTuple = namedtuple('makeTuple', par_dict)
parameters = makeTuple(**par_dict)

#If symbols remain in output below then set them to numerical values in old_par_dict.
parameters._asdict() # print - everything below should have a numeric value
```

Out[15]:

```
{'beta_leaf': 0.21,
 'beta_root': 0.27,
 'r_c_leaf_rh': 0.00993888000000000,
 'r_c_root_rh': 0.0142005600000000,
 'r_c_wood_rh': 0.00299640000000000,
 'r_c_lit_cwd_rh': 0.00353225000000000,
 'r_c_lit_met_rh': 0.00997000000000000,
 'r_c_lit_str_rh': 0.000999800000000000,
 'r_c_lit_mic_rh': 0.0497500000000000,
 'r_c_soil_met_rh': 0.0393600000000000,
 'r_c_soil_str_rh': 0.00389239500000000,
 'r_c_soil_mic_rh': 0.00498000000000000,
 'r_c_soil_slow_rh': 9.99998978286600e-6,
 'r_c_soil_passive_rh': 3.26999893071000e-6,
 'r_c_leaf_2_c_lit_met': 0.00392000000000000,
 'r_c_leaf_2_c_lit_str': 0.000141120000000000,
 'r_c_root_2_c_soil_met': 0.00748000000000000,
 'r_c_root_2_c_soil_str': 0.000319440000000000,
 'r_c_wood_2_c_lit_cwd': 3.60000000000000e-6,
 'r_c_lit_cwd_2_c_lit_mic': 0.00145000000000000,
 'r_c_lit_cwd_2_c_soil_slow': 1.77500000000000e-5,
 'r_c_lit_met_2_c_lit_mic': 3.00000000000000e-5,
 'r_c_lit_str_2_c_lit_mic': 1.00000000000000e-7,
```

```
 'r_c_lit_str_2_c_soil_slow': 1.00000000000000e-7,
 'r_c_lit_mic_2_c_soil_slow': 0.000250000000000000,
 'r_c_soil_met_2_c_soil_mic': 0.000640000000000000,
 'r_c_soil_str_2_c_soil_mic': 4.56300000000000e-6,
 'r_c_soil_str_2_c_soil_slow': 3.04200000000000e-6,
 'r_c_soil_mic_2_c_soil_slow': 1.00000000000000e-5,
 'r_c_soil_mic_2_c_soil_passive': 1.00000000000000e-5,
 'r_c_soil_slow_2_c_soil_mic': 1.00000000000000e-11,
 'r_c_soil_slow_2_c_soil_passive': 2.17134000000000e-13,
 'r_c_soil_passive_2_c_soil_mic': 1.06929000000000e-12}
```

Assign Initial Pool Values

In [16]:

```
# Create vector of initial pool values
svs_0=observables(*map(lambda v: v[0],svs))

# Assign values to initial pools using InitialPools named tuple
V_init = InitialPools(
    c_leaf_0 = svs_0.cVeg/3,          #set inital pool values to svs values
    c_root_0 = svs_0.cVeg/3,          #you can set numerical values here directly as wel
    c_wood_0 = svs_0.cVeg/3,
    c_lit_cwd_0 = svs_0.cSoil/9,
    c_lit_met_0 = svs_0.cSoil/9,
    c_lit_str_0 = svs_0.cSoil/9,
    c_lit_mic_0 = svs_0.cSoil/9,
    c_soil_met_0 = svs_0.cSoil/9,
    c_soil_str_0 = svs_0.cSoil/9,
    c_soil_mic_0 = svs_0.cSoil/9,
    c_soil_slow_0 = svs_0.cSoil/9,
    c_soil_passive_0 = svs_0.cSoil/9
)
V_init._asdict()   #print - everything should have a numeric value
```

Out[16]:

```
{'c_leaf_0': 0.2633374134699504,
 'c_root_0': 0.2633374134699504,
 'c_wood_0': 0.2633374134699504,
 'c_lit_cwd_0': 1.7097499635484483,
 'c_lit_met_0': 1.7097499635484483,
 'c_lit_str_0': 1.7097499635484483,
 'c_lit_mic_0': 1.7097499635484483,
 'c_soil_met_0': 1.7097499635484483,
 'c_soil_str_0': 1.7097499635484483,
 'c_soil_mic_0': 1.7097499635484483,
 'c_soil_slow_0': 1.7097499635484483,
 'c_soil_passive_0': 1.7097499635484483}
```

Define Forward Model

Create constants for forward sim:

In [17]:

```
cpa = Constants(             #use Constants namedtuple to define constant values
    npp_0 = dvs.npp[0],
    rh_0 = svs.rh[0],
    c_veg_0 = svs.cVeg[0],
    c_soil_0 = svs.cSoil[0],
    clay = 0.2028,
    silt = 0.2808,
    nyears = 320
)
cpa._asdict()    #print - everything should have a numeric value
```

Out[17]:

```
{'npp_0': 0.005385355188991525,
 'rh_0': 0.005173560293769697,
```

```
 'c_veg_0': 0.79001224,
 'c_soil_0': 15.38775,
 'clay': 0.2028,
 'silt': 0.2808,
 'nyears': 320}
```

Create list of parameters to be optimized during data assimilation:

In [18]:
```
estimated = {**parameters._asdict(),**V_init._asdict()}            # Create dictionary o
OptimizedParameters = namedtuple('OptimizedParameters', estimated) # Create function to
epa0 = OptimizedParameters(**estimated)                            # Create namedtuple o
epa0._asdict()   #print
```

Out[18]:
```
{'beta_leaf': 0.21,
 'beta_root': 0.27,
 'r_c_leaf_rh': 0.00993888000000000,
 'r_c_root_rh': 0.0142005600000000,
 'r_c_wood_rh': 0.00299640000000000,
 'r_c_lit_cwd_rh': 0.00353225000000000,
 'r_c_lit_met_rh': 0.00997000000000000,
 'r_c_lit_str_rh': 0.000999800000000000,
 'r_c_lit_mic_rh': 0.0497500000000000,
 'r_c_soil_met_rh': 0.0393600000000000,
 'r_c_soil_str_rh': 0.00389239500000000,
 'r_c_soil_mic_rh': 0.00498000000000000,
 'r_c_soil_slow_rh': 9.99998978286600e-6,
 'r_c_soil_passive_rh': 3.26999893071000e-6,
 'r_c_leaf_2_c_lit_met': 0.00392000000000000,
 'r_c_leaf_2_c_lit_str': 0.000141120000000000,
 'r_c_root_2_c_soil_met': 0.00748000000000000,
 'r_c_root_2_c_soil_str': 0.000319440000000000,
 'r_c_wood_2_c_lit_cwd': 3.60000000000000e-6,
 'r_c_lit_cwd_2_c_lit_mic': 0.00145000000000000,
 'r_c_lit_cwd_2_c_soil_slow': 1.77500000000000e-5,
 'r_c_lit_met_2_c_lit_mic': 3.00000000000000e-5,
 'r_c_lit_str_2_c_lit_mic': 1.00000000000000e-7,
 'r_c_lit_str_2_c_soil_slow': 1.00000000000000e-7,
 'r_c_lit_mic_2_c_soil_slow': 0.000250000000000000,
 'r_c_soil_met_2_c_soil_mic': 0.000640000000000000,
 'r_c_soil_str_2_c_soil_mic': 4.56300000000000e-6,
 'r_c_soil_str_2_c_soil_slow': 3.04200000000000e-6,
 'r_c_soil_mic_2_c_soil_slow': 1.00000000000000e-5,
 'r_c_soil_mic_2_c_soil_passive': 1.00000000000000e-5,
 'r_c_soil_slow_2_c_soil_mic': 1.00000000000000e-11,
 'r_c_soil_slow_2_c_soil_passive': 2.17134000000000e-13,
 'r_c_soil_passive_2_c_soil_mic': 1.06929000000000e-12,
 'c_leaf_0': 0.2633374134699504,
 'c_root_0': 0.2633374134699504,
 'c_wood_0': 0.2633374134699504,
 'c_lit_cwd_0': 1.7097499635484483,
 'c_lit_met_0': 1.7097499635484483,
 'c_lit_str_0': 1.7097499635484483,
 'c_lit_mic_0': 1.7097499635484483,
 'c_soil_met_0': 1.7097499635484483,
 'c_soil_str_0': 1.7097499635484483,
 'c_soil_mic_0': 1.7097499635484483,
 'c_soil_slow_0': 1.7097499635484483,
 'c_soil_passive_0': 1.7097499635484483}
```

Create forward model function:

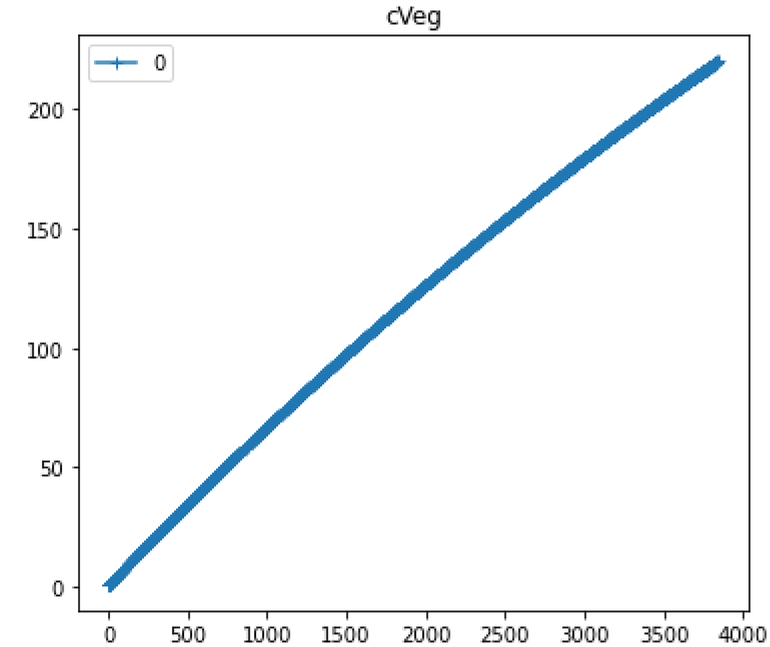
In [19]:
```
def make_param2res_sym(
        cpa: Constants
    ) -> Callable[[np.ndarray], np.ndarray]:
```

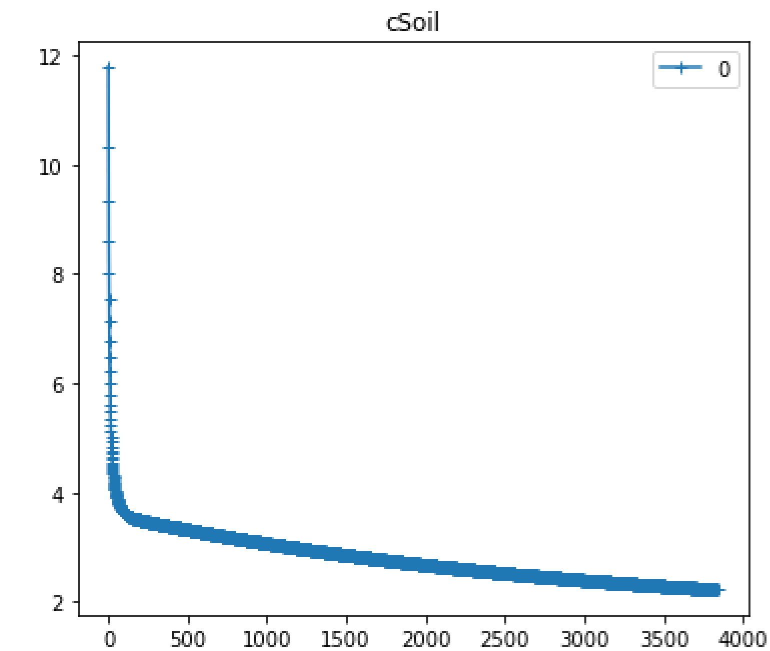
```
    # Build iterator
    # Need dictionary of numeric values for all parameters that are not state variables/
    srm=mvs.get_SmoothReservoirModel()
    model_par_dict_keys=srm.free_symbols.difference(
        [Symbol(str(mvs.get_TimeSymbol()))]+
        list(mvs.get_StateVariableTuple())
    )

    # Create namedtuple function for initial values
    StartVector=namedtuple(
        "StartVector",
            [str(v) for v in mvs.get_StateVariableTuple()]+["rh"]
    )

    # Time dependent driver function does not change with the estimated parameters
    # Defined once outside param2res function
    def npp_func(day):
        month=day_2_month_index(day)
        return dvs.npp[month]

    # Define actual forward simulation function
    def param2res(pa):

        # Parameter vector
        epa=OptimizedParameters(*pa)

        # Create a startvector for the iterator
        V_init = StartVector(
            c_leaf=epa.c_leaf_0,
            c_root=epa.c_root_0,
            c_wood=cpa.c_veg_0-(epa.c_leaf_0 + epa.c_root_0),
            c_lit_cwd=epa.c_lit_cwd_0,
            c_lit_met=epa.c_lit_met_0,
            c_lit_str=epa.c_lit_str_0,
            c_lit_mic=epa.c_lit_mic_0,
            c_soil_met=epa.c_soil_met_0,
            c_soil_str=epa.c_soil_str_0,
            c_soil_mic=epa.c_soil_mic_0,
            c_soil_slow=epa.c_soil_slow_0,
            c_soil_passive=cpa.c_soil_0-(epa.c_lit_cwd_0+epa.c_lit_met_0+epa.c_lit_str_0
            rh=cpa.rh_0
        )

        # Parameter dictionary for the iterator
        apa = {**cpa._asdict(),**epa._asdict()}
        model_par_dict = {
            k:v for k,v in apa.items()
            if k in model_par_dict_keys
        }

        # Build environmental scaler function
        def xi_func(day):
            return 1.0     # Set to 1 if no scaler implemented

        # Define function dictionary
        func_dict={
            'NPP':npp_func,
             'xi':xi_func
        }

        # Construct daily iterator
        def make_daily_iterator_sym(
                mvs,
                V_init: StartVector,
                par_dict,
```

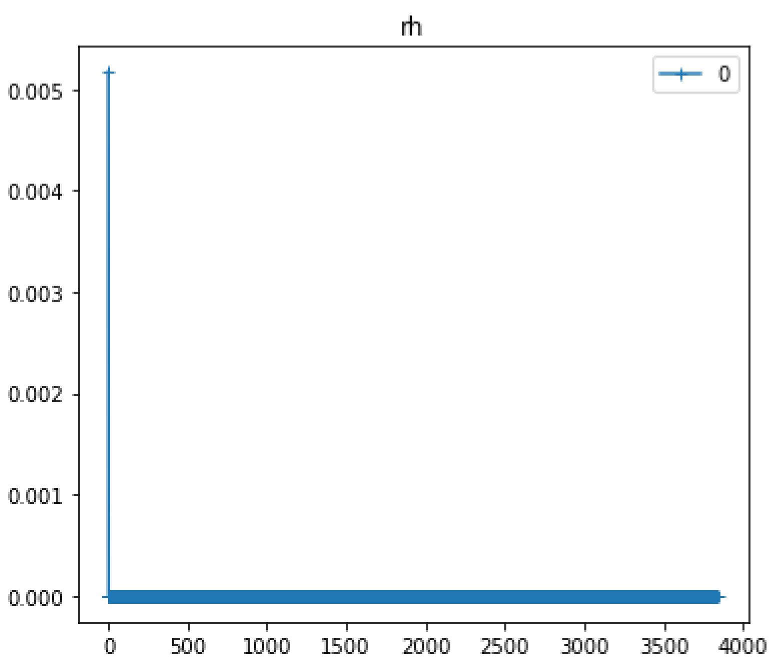
```
                func_dict
            ):

            #
            B_func, u_func = make_B_u_funcs_2(mvs,par_dict,func_dict)
            sv=mvs.get_StateVariableTuple()
            n=len(sv)

            # Create numpy array from named tuple of intial values
            V_arr=np.array(V_init).reshape(n+1,1) #reshaping is neccessary for matmux

            # Define function for matrix math
            def f(it,V):
                X = V[0:n]
                b = u_func(it,X)
                B = B_func(it,X)
                outfluxes = B @ X
                X_new = X + b + outfluxes
                # we also compute the autotrophic and heterotrophic respiration in every
                rh=0
                V_new = np.concatenate((X_new.reshape(n,1),np.array([rh]).reshape(1,1)),
                return V_new

            #
            return TimeStepIterator2(
                initial_values=V_arr,
                f=f,
            )

        # Iterator function
        it_sym = make_daily_iterator_sym(
            mvs,
            V_init=V_init,
            par_dict=par_dict,
            func_dict=func_dict
        )

        # Now that we have the iterator we can start to compute.
        # Note: check if TRENDY months are like this...
        days_per_month = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]

        #empty array for saving data
        sols=[]
        for m in range(cpa.nyears*12):      # Loop through months
            dpm = days_per_month[ m % 12]    # Set days for each month
            mrh=0                            # Start respiration sum at zero each month
            for d in range(dpm):             # Loop through days in month
                v = it_sym.__next__()        # Update initial vector each day
                mrh +=v[12,0]                # Sum respiration
            V=StartVector(*v)                #
            o=observables(                   #
                cVeg=float(V.c_leaf+V.c_wood+V.c_root),
                cSoil=float(V.c_lit_cwd+V.c_lit_met+V.c_lit_str+V.c_lit_mic+ \
                      V.c_soil_met+V.c_soil_str+V.c_soil_mic+V.c_soil_slow+V.c_soi

                rh=mrh,
            )
            sols.append(o) # Append monthly value to results
        sol=np.stack(sols) # Stack arrays
        return sol
    return param2res
```

Forward Model Run

In [20]:
```
const_params = cpa                                   # Define constant parameters
param2res_sym = make_param2res_sym(const_params) # Define forward model
xs= param2res_sym(epa0)                              # Run forward model from initial condit

# Plot simulation output for observables
fig = plt.figure()
plot_solutions(
        fig,
        times=range(cpa.nyears*12),
        var_names=observables._fields,
        tup=(xs,)
)
fig.savefig('solutions.pdf')
```


cVeg
0
200
150
100
50
0
0
500
1000
1500
2000
2500
3000
3500
4000


cSoil
0
12
10
8
6
4
2
0
500
1000
1500
2000
2500
3000
3500
4000


rh
0
0.005
0.004
0.003
0.002
0.001
0.000
0
500
1000
1500
2000
2500
3000
3500
4000

Data Assimilation

Coming soon