

```
#####
#
# Sarah Laskey
# 21 September 2012
# iip.py
# Calculate (combined) IIP over 24-hour dosing interval.
#
#####

#####
#
# IMPORT
#

import math
import pylab

#####
#
# GLOBAL CONSTANTS
#

max_iterations = 100000
cutoff_delta = 0.000001

#####
#
# DRUGS (CLASSES)
#

# this seems lazy because i typed all that stuff in by hand, but also the best
# way to do this.

class Drug:
    """
    this is for keeping track of pharmacokinetic/pharmacodynamic parameters for
    each drug analyzed.
    """

    def __init__(self, m, ic, cmax, halflife, interval):
        """
        instantiate drug with parameters provided. cmax and ic50 are [uM], dosing
        interval and half life are stored as [min].
        """

        self.m = m
        self.ic = ic
        self.cmax = cmax
        self.thalf = halflife * 60
        self.interval = interval * 60

    def set_concentration(self, c):
        """
        set a concentration for this drug. (this is temporary.)
        """
        self.c = c

dtg = Drug(1.3, 0.045, 8.75, 14, 24)
abc = Drug(1.05, 0.0777, 13.45, 18, 24)
lam = Drug(1.13, 0.0557, 15.29, 10, 24)
tdf = Drug(0.42, 0.013, 1.06, 180, 24)
ftc = Drug(1.22, 0.0284, 6.8, 39, 24)
ral = Drug(0.91, 0.0418, 4.5, 9.5, 12)

#####
#
# METHODS
#

def iip(drugs, di):
```

```

'''
given one or more drugs, calculate their combined IIP at short intervals over
a 24-hour dosing period.  return the whole time course, i guess?
'''

# loewe, bliss, and expected (using DI) combined drug efficacy
iip_L = []
iip_B = []
iip_E = []

# 1440 minutes in a day...
for t in range(1440):

    # calculate and set the [drug] for each drug at this time point
    for d in drugs:

        # time since last dosing
        dose_t = t % (d.interval * 60)
        d.set_concentration(d.cmax * math.pow(2, -1.0 * dose_t / d.thalf))

    # calculate log(fa/fu) for loewe, bliss, and (eventually) DI midpoint
    m = [d.m for d in drugs]
    ic = [d.ic for d in drugs]
    c = [d.c for d in drugs]

    f_L = loewe(m, ic, c)
    f_B = bliss(m, ic, c)
    f_E = f_L + di * (f_B - f_L)

    # calculate IIP
    fu_L = 1.0 / (1.0 + math.pow(10, f_L))
    fu_B = 1.0 / (1.0 + math.pow(10, f_B))
    fu_E = 1.0 / (1.0 + math.pow(10, f_E))

    iip_L += [math.log(1.0/fu_L, 10)]
    iip_B += [math.log(1.0/fu_B, 10)]
    iip_E += [math.log(1.0/fu_E, 10)]

    if t == 0: print 'cmax', c
    if t == 1439: print 'cmin', c

return iip_L, iip_B, iip_E

def bliss(m, ic, d):
    '''
    calculates the Loewe log(fa/fu) value, given lists of (equivalently indexed)
    slopes, ic50s, and drug concentrations.
    '''
    # well, this is easy.

    f = 1

    for i in range(len(m)):
        temp = 1 / (1.0 + math.pow(d[i]/ic[i], m[i]))
        f *= temp

    return math.log10((1-f)/f)

def loewe(m, ic, d):
    '''
    calculates the Loewe log(fa/fu) value, given lists of (equivalently indexed)
    slopes, ic50s, and drug concentrations.
    '''

    if not len(m)==len(ic):
        print 'something is wrong.'
        raise Exception

    # initiate markers that will converge
    upper = 5.0
    lower = -15.0

```

```

# keep track of progress
delta = cutoff_delta + 1
iterations = 0

# iterate
while delta > cutoff_delta and iterations < max_iterations:

    iterations += 1

    # calculate upper y
    upper_x = math.pow(10, upper)
    upper_y = -1.0 + sum([d[a] / (ic[a] * math.pow(upper_x, -1.0/m[a]))
                        for a,b in enumerate(m)])

    # calculate lower y
    lower_x = math.pow(10, lower)
    lower_y = -1.0 + sum([d[a] / (ic[a] * math.pow(lower_x, -1.0/m[a]))
                        for a,b in enumerate(m)])

    # break program if answer isn't within range explored
    if (upper_y > 0 and lower_y > 0) or (upper_y < 0 and lower_y < 0):
        print 'the range of fa/fu values explored does not include the solution'
        raise Exception

    # calculate middle y
    middle = (upper + lower) / 2
    middle_x = math.pow(10, middle)
    middle_y = -1.0 + sum([d[a] / (ic[a] * math.pow(middle_x, -1.0/m[a]))
                        for a,b in enumerate(m)])

    if middle_y > 0: upper = middle
    if middle_y < 0: lower = middle

    if math.fabs(upper - lower) < cutoff_delta:
        #print 'solution generated in', str(iterations), 'iterations.'
        return (upper+lower) / -2.0

    if iterations >= max_iterations:
        print 'program exited after', str(iterations), 'iterations.'
        print 'upper =', str(upper)
        print 'lower =', str(lower)
        return (upper+lower) / -2.0

#####
#
# EXECUTION
#

l, b, e = iip([dtg, lam, abc], 0.858460858)

print 'DTG/3TC/ABC'
print 'L', sum(l)/len(l), max(l), min(l)
print 'B', sum(b)/len(b), max(b), min(b)
print 'E', sum(e)/len(e), max(e), min(e)
print

f = e

l, b, e = iip([dtg, tdf, ftc], 1.477690485)

print 'DTG/TDF/FTC'
print 'L', sum(l)/len(l), max(l), min(l)
print 'B', sum(b)/len(b), max(b), min(b)
print 'E', sum(e)/len(e), max(e), min(e)
print

#pylab.plot(range(1440), [5]*1440)
#pylab.plot(e)
#pylab.show()

l, b, e = iip([dtg], 1)

```

```

print 'DTG alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([abc], 1)
print 'ABC alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([lam], 1)
print '3TC alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([tdf], 1)
print 'TDF alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([ftc], 1)
print 'FTC alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([ral], 1)
print 'RAL alone'
print 'E', sum(e)/len(e), max(e), min(e)
print

l, b, e = iip([dtg, lam], 1.184991827)
print 'DTG + 3TC'
print 'L', sum(l)/len(l), max(l), min(l)
print 'B', sum(b)/len(b), max(b), min(b)
print 'E', sum(e)/len(e), max(e), min(e)
print

```