

Sep 23, 14 13:55		csc710sbse: HW4:Theisen				Page 1/2
-- *		0.26151,	0.31919,	0.35434,	0.52981,	0.64670
-- *		0.21715,	0.26667,	0.29704,	0.46117,	0.61612
-- *		0.28271,	0.35877,	0.38036,	0.39697,	0.42963
-- *		0.49841,	0.52073,	0.53804,	0.55058,	0.56790
-- *		0.48495,	0.51691,	0.55418,	0.55961,	0.58869
Model Name: ZDT3						
Searcher Name: MaxWalkSat						
Seed: 1						
MaxWalkSat Options:						
Prob: 0.75						
MaxTries: 500 MaxChanges 500						
Threshold: 1e-06 Slices: 10						
Time to run (s): 0.319793						
Runs: 10						
Average per run (s): 0.0319793						
*		-0.00536,	-0.00536,	-0.00536,	-0.00536,	-0.00536

- *		0.00379,	0.00459,	0.00713,	0.00753,	0.01312
- *		0.00409,	0.00492,	0.00728,	0.01077,	0.01338
- *		0.00409,	0.00641,	0.00735,	0.00876,	0.01177
- *		0.00409,	0.00611,	0.00783,	0.01127,	0.01879
- *		0.00409,	0.00669,	0.00843,	0.01160,	0.01601
- *		0.00399,	0.00728,	0.00900,	0.01855,	0.02151
- *		0.00409,	0.00547,	0.00651,	0.00714,	0.01339
- *		0.00409,	0.00608,	0.00791,	0.01537,	0.02215
- *		0.00522,	0.00661,	0.00767,	0.00981,	0.01614
- *		0.00458,	0.00695,	0.00776,	0.00954,	0.01383
- *		0.00382,	0.00532,	0.00753,	0.00944,	0.01623
- *		0.00409,	0.00468,	0.00694,	0.00739,	0.00827
- *		0.00409,	0.00409,	0.00641,	0.01191,	0.02317
- *		0.00409,	0.00617,	0.00780,	0.01223,	0.02425
- *		0.00407,	0.00511,	0.00686,	0.00809,	0.01101
- *		0.00433,	0.00566,	0.00738,	0.00997,	0.02194
- *		0.00535,	0.00720,	0.00851,	0.01097,	0.01894
- *		0.00409,	0.00496,	0.00646,	0.00988,	0.01383
- *		0.00499,	0.00789,	0.00864,	0.01215,	0.02221
- *		0.00409,	0.00657,	0.00782,	0.01158,	0.01487
- *		0.00412,	0.00600,	0.00751,	0.00922,	0.01968
- *		0.00452,	0.00681,	0.00841,	0.01032,	0.02064
- *		0.00482,	0.00660,	0.00715,	0.00823,	0.01186
- *		0.00409,	0.00713,	0.00951,	0.01204,	0.01392
- *		0.00467,	0.00656,	0.00851,	0.01320,	0.01654
- *		0.00442,	0.00559,	0.00752,	0.01284,	0.02053
- *		0.00409,	0.00605,	0.00741,	0.00960,	0.02046
- *		0.00409,	0.00756,	0.00864,	0.01129,	0.01993
- *		0.00409,	0.00462,	0.00643,	0.00819,	0.00987
- *		0.00428,	0.00589,	0.00910,	0.01063,	0.01788
- *		0.00377,	0.00458,	0.00584,	0.01037,	0.02271
- *		0.00383,	0.00445,	0.00696,	0.00954,	0.01068
- *		0.00409,	0.00757,	0.00904,	0.01426,	0.02366
- *		0.00625,	0.00795,	0.00977,	0.01180,	0.02071
- *		0.00377,	0.00672,	0.00748,	0.00961,	0.01576
- *		0.00409,	0.00537,	0.00952,	0.01005,	0.02346
- *		0.00456,	0.00584,	0.00723,	0.00910,	0.01676
- *		0.00551,	0.00697,	0.00859,	0.01414,	0.02305
- *		0.00409,	0.00501,	0.00742,	0.00882,	0.02402
- *		0.00409,	0.00509,	0.00625,	0.00744,	0.00886
- *		0.00448,	0.00740,	0.00814,	0.00961,	0.01400
- *		0.00441,	0.00536,	0.00728,	0.00818,	0.01303
- *		0.00409,	0.00695,	0.00802,	0.01429,	0.02147
- *		0.00491,	0.00671,	0.00923,	0.01192,	0.02005
- *		0.00364,	0.00409,	0.00534,	0.00926,	0.01046
- *		0.00409,	0.00754,	0.00911,	0.01369,	0.02138
- *		0.00409,	0.00550,	0.00721,	0.00831,	0.01099
- *		0.00409,	0.00675,	0.00768,	0.01276,	0.01770
- *		0.00409,	0.00597,	0.00901,	0.01241,	0.01663
- *		0.00418,	0.00692,	0.00847,	0.01151,	0.01344
- *		0.00409,	0.00711,	0.00857,	0.01122,	0.02057
- *		0.00466,	0.00655,	0.00843,	0.01339,	0.02301
- *		0.00409,	0.00689,	0.00795,	0.01026,	0.01993
- *		0.00493,	0.00605,	0.00895,	0.00983,	0.02039
- *		0.00409,	0.00737,	0.01027,	0.01343,	0.02333
- *		0.00409,	0.00463,	0.00654,	0.00754,	0.01073
- *		0.00504,	0.00659,	0.00755,	0.01177,	0.01719
- *		0.00250,	0.00543,	0.00746,	0.00916,	0.01269
- *		0.00409,	0.00662,	0.00713,	0.00883,	0.01336
- *		0.00358,	0.00584,	0.00639,	0.01160,	0.02331
- *		0.00409,	0.00462,	0.00767,	0.01305,	0.02444
- *		0.00612,	0.00712,	0.00903,	0.01172,	0.01793
- *		0.00224,	0.00583,	0.00710,	0.00884,	0.01776
- *		0.00454,	0.00739,	0.01230,	0.01454,	0.02107
- *		0.00409,	0.00626,	0.00898,	0.01218,	0.02347
- *		0.00483,	0.00624,	0.00706,	0.00813,	0.02204
- *		0.00476,	0.00538,	0.01107,	0.01592,	0.02334
- *		0.00359,	0.00643,	0.00781,	0.01115,	0.02132
- *		0.00473,	0.00745,	0.00870,	0.01372,	0.02352
- *		0.00218,	0.00528,	0.00774,	0.00893,	0.01310
- *		0.00409,	0.00563,	0.00708,	0.00907,	0.01675
- *		0.00388,	0.00514,	0.00826,	0.01313,	0.02281
- *		0.00437,	0.00600,	0.00815,	0.01044,	0.01646
- *		0.00391,	0.00506,	0.00585,	0.00771,	0.01302
- *		0.00457,	0.00671,	0.00701,	0.01037,	0.01897
- *		0.00467,	0.00702,	0.00860,	0.01256,	0.02216
- *		0.00472,	0.00745,	0.00959,	0.01315,	0.02029

Sep 23, 14 13:55		csc710sbse: HW4:Theisen				Page 2/2
-- *		0.00453,	0.00914,	0.01260,	0.01801,	0.02511
-- *		0.00508,	0.00765,	0.01210,	0.01793,	0.02295
-- *		0.00462,	0.00798,	0.01110,	0.01467,	0.02146
-- *		0.00273,	0.00581,	0.00702,	0.01067,	0.02064
-- *		0.00409,	0.00712,	0.00740,	0.01004,	0.01338
-- *		0.00409,	0.00606,	0.00720,	0.00814,	0.01216
-- *		0.00409,	0.00606,	0.00786,	0.01067,	0.01620
-- *		0.00509,	0.00683,	0.00846,	0.01203,	0.02180
-- *		0.00409,	0.00591,	0.00795,	0.00989,	0.02187
-- *		0.00412,	0.00525,	0.00765,	0.01257,	0.02294
-- *		0.00450,	0.00731,	0.00880,	0.01186,	0.01832
-- *		0.00409,	0.00602,	0.00714,	0.00811,	0.02165
-- *		0.00409,	0.00714,	0.00889,	0.01223,	0.01566
-- *		0.00409,	0.00685,	0.00979,	0.01131,	0.02236
-- *		0.00446,	0.00653,	0.00972,	0.00952,	0.01330
-- *		0.00409,	0.00665,	0.00766,	0.01311,	0.02095
-- *		0.00458,	0.00548,	0.00678,	0.00771,	0.01270
-- *		0.00460,	0.00653,	0.00747,	0.00930,	0.02011
-- *		0.00462,	0.00699,	0.00743,	0.00952,	0.02044
-- *		0.00409,	0.00669,	0.00739,	0.00805,	0.01115
-- *		0.00634,	0.00724,	0.00994,	0.01322,	0.02318
-- *		0.00409,	0.00497,	0.00679,	0.00789,	0.01637
-- *		0.00459,	0.00628,	0.01043,	0.01311,	0.02367
-- *		0.00409,	0.00839,	0.01000,	0.01300,	0.02297
-- *		0.00462,	0.00653,	0.00820,	0.01220,	0.01993
-- *		0.00620,	0.00725,	0.01055,	0.01471,	0.01860
-- *		0.00465,	0.00590,	0.00734,	0.01012,	0.01819
-- *		0.00401,	0.00688,	0.00870,	0.01123,	0.01682
-- *		0.00409,	0.00458,	0.00675,	0.00961,	0.01319
-- *		0.00409,	0.00722,	0.00770,	0.00976,	0.02224
-- *		0.00471,	0.00733,	0.00895,	0.01066,	0.01823
-- *		0.00357,	0.00487,	0.00750,	0.00985,	0.01322
-- *		0.00352,	0.00461,	0.00672,	0.00991,	0.01958
-- *		0.00445,	0.00924,	0.01571,	0.02010,	0.02385
-- *		0.00409,	0.00662,	0.00957,	0.01537,	0.02386
-- *		0.00291,	0.00464,	0.00641,	0.00728,	0.00918
-- *		0.00206,	0.00549,	0.00739,	0.00951,	0.02304
-- *		0.00265,	0.00528,	0.00781,	0.01371,	0.02100
-- *		0.00409,	0.00666,	0.00741,	0.00964,	0.01578
-- *		0.00409,	0.00757,	0.00956,	0.01443,	0.02342

Sep 23, 14 14:01

csc710sbse: HW4:Theisen

Page 1/1

```

rank ,      name ,      med      , iqr
-----
1 ,      x1 ,      51      , 11 ( *
2 ,      x2 ,      800     , 200 (
-----
rank ,      name ,      med      , iqr
-----
1 ,      x1 ,      30      , 20 ( *
1 ,      x2 ,      30      , 20 ( *
2 ,      x3 ,      800     , 200 (
-----
rank ,      name ,      med      , iqr
-----
1 ,      x5 ,      30      , 20 ( --- * ---
1 ,      x3 ,      35      , 15 ( --- * ---
2 ,      x1 ,      51      , 11 ( --- * ---
3 ,      x2 ,      80      , 20 (
3 ,      x4 ,      80      , 20 (
-----
rank ,      name ,      med      , iqr
-----
1 ,      x1 ,     10000    , 150 (----- *
1 ,      x2 ,     10000    , 100 (----- *
1 ,      x3 ,     10000    , 150 (----- *
1 ,      x4 ,     10000    , 100 (----- *
-----
rank ,      name ,      med      , iqr
-----
1 ,      x1 ,     1100    , 0 ( *
1 ,      x2 ,     1400    , 0 ( *
2 ,      x3 ,     2300    , 0 (
2 ,      x5 ,     3200    , 0 (
-----
rank ,      name ,      med      , iqr
-----
1 ,      x1 ,     1100    , 0 ( *
1 ,      x2 ,     1100    , 0 ( *
1 ,      x3 ,     1100    , 0 ( *
-----
rank ,      name ,      med      , iqr
-----
1 ,      x1 ,     1100    , 0 ( *
1 ,      x2 ,     1100    , 0 ( *
2 ,      x4 ,     3400    , 200 (
-----
rank ,      name ,      med      , iqr
-----
1 ,      x2 ,      24      , 54 ( -- *
2 ,      x3 ,      51      , 49 ( --- *
3 ,      x1 ,      71      , 38 ( --- *
-----
rank ,      name ,      med      , iqr
-----
1 ,      kids ,     2000    , 300 ( *
2 , cookbooks ,     7800    , 12700 ( - * --
2 , magazine ,     8200    , 3500 ( - *
3 , novels ,     44300    , 20200 ( -- | *

```

Sep 23, 14 13:35

csc710sbse: HW4:Theisen

Page 1/1

```
#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

class Options:
    #Globals
    debug = False
    seed = 1

    #MaxWalkSat options
    mws_prob = 0.75
    mws_maxTries = 500
    mws_maxChanges = 500
    mws_threshold = .000001
    mws_slices = 10

    #Simulated Annealing options
    sa_kmax = 500
    sa_cooling = .6

    def printGlobals(self):
        print "Seed:", self.seed
```

Sep 23, 14 13:25

csc710sbse: HW4:Theisen

Page 1/1

```
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from sk import *

def rdiv8():
    rdivDemo([
        ["novels", 287, 332, 443, 711, 534],
        ["kids", 23, 18, 16, 20, 21],
        ["magazine", 112, 98, 43, 63, 82],
        ["cookbooks", 232, 180, 32, 53, 78],
    ])

rdiv8()
```

Sep 23, 14 13:49

csc710sbse: HW4:Theisen

Page 1/1

```

import sys
from datetime import datetime
import random

sys.dont_write_bytecode = True

from models import *
from searchers import *
from utils import *
from options import *

myOpt = Options()

#Inspired by vivekaxl's display function
def display(model, searcher, startTime, scores, r):
    print "=====
    print "Model Name: ", model.__name__
    print "Searcher Name: ", searcher.name
    diff = (datetime.now() - startTime).total_seconds()
    myOpt.printGlobals()
    searcher.printOptions()
    print "Time to run (s): ", diff
    if r == 0:
        print "No valid runs!"
    else:
        print "Runs: ", r
        print "Average per run (s): ", diff/r
        print xtile(scores,width=25,show=" %1.5f")
    print "=====

def main(modelList, searcherList):
    r = 10
    for klass in modelList:
        for searcher in searcherList:
            startTime = datetime.now()
            scores = []
            myKlass = klass()
            mySearcher = searcher()
            random.seed(myOpt.seed)
            for _ in range(r):
                result, valid = mySearcher.run(myKlass)
                if valid == True:
                    scores.append(result)
            display(klass, mySearcher, startTime, scores, len(scores))

modelList = [ZDT3, Viennet3]
searcherList = [MWS]

main(modelList, searcherList)

```

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 1/7

```

"""
## Hypothesis Testing Stuff

### Standard Stuff

#### Standard Headers

"""
from __future__ import division
import sys, random, math
sys.dont_write_bytecode = True
"""

#### Standard Utils

"""
class o():
    "Anonymous container"
    def __init__(i,**fields) :
        i.override(fields)
    def override(i,d): i.__dict__.update(d); return i
    def __repr__(i):
        d = i.__dict__
        name = i.__class__.__name__
        return name+'{'+' '.join(['%s%s' % (k,pretty(d[k]))
                                for k in i.show()])+' '}'
    def show(i):
        return [k for k in sorted(i.__dict__.keys())
                if not k in i.__dict__['_']]
"""

Misc functions:

"""
rand = random.random
any = random.choice
seed = random.seed
exp = lambda n: math.e**n
ln = lambda n: math.log(n,math.e)
g = lambda n: round(n,2)

def median(lst,ordered=False):
    if not ordered: lst= sorted(lst)
    n = len(lst)
    p = n//2
    if n % 2: return lst[p]
    q = p - 1
    q = max(0,min(q,n))
    return (lst[p] + lst[q])/2

def msec(f):
    import time
    t1 = time.time()
    f()
    return (time.time() - t1) * 1000

def pairs(lst):
    "Return all pairs of items i,i+1 from a list."
    last=lst[0]
    for i in lst[1:]:
        yield last,i
        last = i

def xtile(lst,lo=0,hi=100,width=50,
          chops=[0.1 ,0.3,0.5,0.7,0.9],
          marks=["-", " ", " ", " ", "-"],
          bar="|",star="*",show="%3.0f"):
    """The function _xtile_ takes a list of (possibly)
    unsorted numbers and presents them as a horizontal
    xtile chart (in ascii format). The default is a
    contracted quintile_ that shows the
    10,30,50,70,90 breaks in the data (but this can be
    changed-- see the optional flags of the function).
    """
    def pos(p) : return ordered[int(len(lst)*p)]
    def place(x) :
        return int(width*float((x - lo))/(hi - lo+0.00001))
    def pretty(lst) :
        return ' '.join([show % x for x in lst])
    ordered = sorted(lst)
    lo = min(lo,ordered[0])
    hi = max(hi,ordered[-1])
    what = [pos(p) for p in chops]
    where = [place(n) for n in what]
    out = [" "] * width
    for one,two in pairs(where):
        for i in range(one,two):
            out[i] = marks[0]
        marks = marks[1:]
    out[int(width/2)] = bar
    out[place(pos(0.5))] = star
    return '({'+''.join(out) + "},)" + pretty(what)

```

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 2/7

```

def _xtileX() :
    import random
    random.seed(1)
    nums = [random.random()*2 for _ in range(100)]
    print xtile(nums,lo=0,hi=1.0,width=25,show=" %5.2f")
"""

### Standard Accumulator for Numbers

Note the _lt_ method: this accumulator can be sorted by median values.

Warning: this accumulator keeps _all_ numbers. Might be better to use
a bounded cache.

"""
class Num:
    "An Accumulator for numbers"
    def __init__(i,name,init=[1]):
        i.n = i.m2 = i.mu = 0.0
        i.all=[]
        i._median=None
        i.name = name
        i.rank = 0
        for x in init: i.add(x)
    def s(i) : return (i.m2/(i.n - 1))*0.5
    def add(i,x):
        i._median=None
        i.n += 1
        i.all += [x]
        delta = x - i.mu
        i.mu += delta*1.0/i.n
        i.m2 += delta*(x - i.mu)
    def __add__(i,j):
        return Num(i.name + j.name,i.all + j.all)
    def quartiles(i):
        def p(x) : return int(100*g(xs[x]))
        i.median()
        xs = i.all
        n = int(len(xs)*0.25)
        return p(n) , p(2*n) , p(3*n)
    def median(i):
        if not i._median:
            i.all = sorted(i.all)
            i._median=median(i.all)
        return i._median
    def __lt__(i,j):
        return i.median() < j.median()
    def spread(i):
        i.all=sorted(i.all)
        n1=i.n*0.25
        n2=i.n*0.75
        if len(i.all) <= 1:
            return 0
        if len(i.all) == 2:
            return i.all[1] - i.all[0]
        else:
            return i.all[int(n2)] - i.all[int(n1)]

"""

### The A12 Effect Size Test

"""
def al2slow(lst1,lst2):
    "how often is x in lst1 more than y in lst2?"
    more = same = 0.0
    for x in lst1:
        for y in lst2:
            if x == y : same += 1
            elif x > y : more += 1
    x= (more + 0.5*same) / (len(lst1)*len(lst2))
    return x

def al2(lst1,lst2):
    "how often is x in lst1 more than y in lst2?"
    def loop(t,t1,t2):
        while t1.j < t1.n ^ t2.j < t2.n:
            h1 = t1.l[t1.j]
            h2 = t2.l[t2.j]
            h3 = t2.l[t2.j+1] if t2.j+1 < t2.n else None
            if h1> h2:
                t1.j += 1; t1.gt += t2.n - t2.j
            elif h1 == h2:
                if h3 ^ h1 > h3 :
                    t1.gt += t2.n - t2.j - 1
                    t1.j += 1; t1.eq += 1; t2.eq += 1
            else:
                t2,t1 = t1,t2
        return t.gt*1.0, t.eq*1.0
    #-----
    lst1 = sorted(lst1, reverse=True)
    lst2 = sorted(lst2, reverse=True)
    n1 = len(lst1)
    n2 = len(lst2)

```

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 3/7

```

t1 = o(1=1st1,j=0,eq=0,gt=0,n=n1)
t2 = o(1=1st2,j=0,eq=0,gt=0,n=n2)
gt,eq= loop(t1, t1, t2)
return gt/(n1*n2) + eq/2/(n1*n2)

def _a12():
def f1(): return a12slow(l1,l2)
def f2(): return a12(l1,l2)
for n in [100,200,400,800,1600,3200,6400]:
    l1 = [rand() for _ in xrange(n)]
    l2 = [rand() for _ in xrange(n)]
    t1 = msec(f1)
    t2 = msec(f2)
    print n, g(f1()),g(f2()),int((t1/t2))

```

***Output:

....

n	a12(fast)	a12(slow)	tfast / tslow
100	0.53	0.53	4
200	0.48	0.48	6
400	0.49	0.49	28
800	0.5	0.5	26
1600	0.51	0.51	72
3200	0.49	0.49	109
6400	0.5	0.5	244

100 0.53 0.53 4

200 0.48 0.48 6

400 0.49 0.49 28

800 0.5 0.5 26

1600 0.51 0.51 72

3200 0.49 0.49 109

6400 0.5 0.5 244

....

Non-Parametric Hypothesis Testing

The following `_bootstrap_` method was introduced in 1979 by Bradley Efron at Stanford University. It was inspired by earlier work on the jackknife. Improved estimates of the variance were [developed later][efron01].

[efron01]: <http://goo.gl/14n8WF> "Bradley Efron & R.J. Tibshirani. An Introduction to the Bootstrap (Chapman & Hall/CRC M

To check if two populations `(y0,z0)_` are different, many times sample with replacement from both to generate `(y1,z1)`, `(y2,z2)`, `(y3,z3)_...` etc.

```

***
def sampleWithReplacement(lst):
    "returns a list same size as lst"
    def any(n): return random.uniform(0,n)
    def one(lst): return lst[ int(any(len(lst))) ]
    return [one(lst) for _ in lst]
***

```

Then, for all those samples, check if some `*testStatistic*` in the original pair hold for all the other pairs. If it does more than (say) 99% of the time, then we are 99% confident in that the populations are the same.

In such a `_bootstrap_` hypothesis test, the `*some property*` is the difference between the two populations, muted by the joint standard deviation of the populations.

```

***
def testStatistic(y,z):
    """Checks if two means are different, tempered
    by the sample size of 'y' and 'z'"""
    tmp1 = tmp2 = 0
    for y1 in y.all: tmp1 += (y1 - y.mu)**2
    for z1 in z.all: tmp2 += (z1 - z.mu)**2
    s1 = (float(tmp1)/(y.n - 1))**0.5
    s2 = (float(tmp2)/(z.n - 1))**0.5
    delta = z.mu - y.mu
    if s1+s2:
        delta = delta/((s1/y.n + s2/z.n)**0.5)
    return delta
***

```

The rest is just details:

- + Efron advises to make the mean of the populations the same (see the `_yhat,zhat_` stuff shown below).
- + The class `_total_` is a just a quick and dirty accumulation class.
- + For more details see [the Efron text][efron01].

```

***
def bootstrap(y0,z0,conf=0.01,b=1000):
    """The bootstrap hypothesis test from
    p220 to 223 of Efron's book 'An
    introduction to the bootstrap.'"""
    class total():
        "quick and dirty data collector"

```

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 4/7

```

def __init__(i,some=[]):
    i.sum = i.n = i.mu = 0 ; i.all=[]
    for one in some: i.put(one)
    def put(i,x):
        i.all.append(x);
        i.sum +=x; i.n += 1; i.mu = float(i.sum)/i.n
    def _add_(i1,i2): return total(i1.all + i2.all)
y, z = total(y0), total(z0)
x = y + z
tobs = testStatistic(y,z)
yhat = [y1 - y.mu + x.mu for y1 in y.all]
zhat = [z1 - z.mu + x.mu for z1 in z.all]
bigger = 0.0
for i in range(b):
    if testStatistic(total(sampleWithReplacement(yhat)),
                        total(sampleWithReplacement(zhat))) > tobs:
        bigger += 1
return bigger / b < conf
***

```

Examples

```

***
def _bootstraped():
def worker(n=1000,
           mu1=10, sigmal=1,
           mu2=10.2, sigma2=1):
    def g(mu,sigma): return random.gauss(mu,sigma)
    x = [g(mu1,sigmal) for i in range(n)]
    y = [g(mu2,sigma2) for i in range(n)]
    return n,mu1,sigmal,mu2,sigma2,\
        'different' if bootstrap(x,y) else 'same'
# very different means, same std
print worker(mu1=10, sigmal=10,
             mu2=100, sigma2=10)
# similar means and std
print worker(mu1= 10.1, sigmal=1,
             mu2= 10.2, sigma2=1)
# slightly different means, same std
print worker(mu1= 10.1, sigmal= 1,
             mu2= 10.8, sigma2= 1)
# different in mu eater by large std
print worker(mu1= 10.1, sigmal= 10,
             mu2= 10.8, sigma2= 1)
***

```

Output:

```

....
_bootstraped()

(1000, 10, 10, 100, 10, 'different')
(1000, 10.1, 1, 10.2, 1, 'same')
(1000, 10.1, 1, 10.8, 1, 'different')
(1000, 10.1, 10, 10.8, 1, 'same')
....

```

Warning-- the above took 8 seconds to generate since we used 1000 bootstraps. As to how many bootstraps are enough, that depends on the data. There are results saying 200 to 400 are enough but, since I am suspicious man, I run it for 1000.

Which means the runtimes associated with bootstrapping is a significant issue. To reduce that runtime, I avoid things like an all-pairs comparison of all treatments (see below: Scott-knott). Also, BEFORE I do the bootstrap, I first run the effect size test (and only go to bootstrapping in effect size passes:

```

***
def different(l1,l2):
    #return bootstrap(l1,l2) and a12(l2,l1)
    return a12(l2,l1) ^ bootstrap(l1,l2)
***

```

Saner Hypothesis Testing

The following code, which you should use verbatim does the following:

- + All treatments are clustered into `_ranks_`. In practice, dozens of treatments end up generating just a handful of ranks.
- + The numbers of calls to the hypothesis tests are minimized:
 - + Treatments are sorted by their median value.
 - + Treatments are divided into two groups such that the expected value of the mean values `_after_` the split is minimized;
- + Hypothesis tests are called to test if the two groups are truly difference.
 - + All hypothesis tests are non-parametric and include (1) effect size tests and (2) tests for statistically significant numbers;
 - + Slow bootstraps are executed if the faster `_A12_` tests are passed;

In practice, this means that the hypothesis tests (with confidence of say, 95%) are called on only a logarithmic number of times. So...

- + With this method, 16 treatments can be studied using less than $\sum_{i=1}^{16} \log_2 i = 15$ hypothesis tests and confidence $0.99 < \sup$
- + But if did this with the 120 all-pairs comparisons of the 16 treatments, we would have total confidence $0.99 < \sup_{120} > 0.30$.

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 5/7

For examples on using this code, see `_rdivDemo_` (below).

```
"""
def scottknot(data,cohen=0.3,small=3, useA12=False,epsilon=0.01):
    """Recursively split data, maximizing delta of
    the expected value of the mean before and
    after the splits.
    Reject splits with under 3 items"""
    all = reduce(lambda x,y:x+y,data)
    same = lambda l,r: abs(l.median() - r.median()) <= all.s()*cohen
    if useA12:
        same = lambda l, r: ~ different(l.all,r.all)
    big = lambda n: n > small
    return rdiv(data,all,minMu,big,same,epsilon)
```

```
def rdiv(data, # a list of class Nums
         all, # all the data combined into one num
         div, # function: find the best split
         big, # function: rejects small splits
         same, # function: rejects similar splits
         epsilon): # small enough to split two parts
    """Looks for ways to split sorted data.
    Recurses into each split. Assigns a 'rank' number
    to all the leaf splits found in this way.
    """
```

```
def recurse(parts,all,rank=0):
    "Split, then recurse on each part."
    cut,left,right = maybeIgnore(div(parts,all,big,epsilon),
                                same,parts)

    if cut:
        # if cut, rank "right" higher than "left"
        rank = recurse(parts[:cut],left,rank) + 1
        rank = recurse(parts[cut:],right,rank)
    else:
        # if no cut, then all get same rank
        for part in parts:
            part.rank = rank
    return rank
recurse(sorted(data),all)
return data

def maybeIgnore((cut,left,right), same,parts):
    if cut:
        if same(sum(parts[:cut],Num('upto')),
                sum(parts[cut:],Num('above'))):
            cut = left = right = None
    return cut,left,right
```

```
def minMu(parts,all,big,epsilon):
    """Find a cut in the parts that maximizes
    the expected value of the difference in
    the mean before and after the cut.
    Reject splits that are insignificantly
    different or that generate very small subsets.
    """
    cut,left,right = None,None,None
    before,mu = 0,all.mu
    for i,l,r in leftRight(parts,epsilon):
        if big(l.n) ^ big(r.n):
            n = all.n * 1.0
            now = l.n/n*(mu- l.mu)**2 + r.n/n*(mu- r.mu)**2
            if now > before:
                before,cut,left,right = now,i,l,r
    return cut,left,right
```

```
def leftRight(parts,epsilon=0.01):
    """Iterator. For all items in 'parts',
    return everything to the left and everything
    from here to the end. For reasons of
    efficiency, take a first pass over the data
    to pre-compute and cache right-hand-sides
    """
```

```
rights = {}
n = j = len(parts) - 1
while j > 0:
    rights[j] = parts[j]
    if j < n: rights[j] += rights[j+1]
    j -= 1
left = parts[0]
for i,one in enumerate(parts):
    if i > 0:
        if parts[i].median - parts[i-1].median > epsilon:
            yield i,left,rights[i]
            left += one
    """
```

Putting it All Together

Driver for the demos:

```
"""
def rdivDemo(data):
    def z(x):
        return int(100 * (x - lo) / (hi - lo + 0.00001))
    data = map(lambda lst:Num(lst[0],lst[1:]),
```

Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 6/7

```
data)
print ""
ranks=[]
for x in scottknot(data,useA12=True):
    ranks += [(x.rank,x.median(),x)]
all=[]
for _,x in sorted(ranks): all += x.all
all = sorted(all)
lo, hi = all[0], all[-1]
line = "-----"
last = None
print ('%4s,%8s, %s,%4s' % \
      ('rank', 'name', 'med', 'iqr'))+ "\n" + line
for _,x in sorted(ranks):
    q1,q2,q3 = x.quartiles()
    print ('%4s,%8s, %4s,%4s' % \
          (x.rank+1, x.name, q2, q3 - q1)) + \
          xtile(x.all,lo=lo,hi=hi,width=30,show="%5.2f")
    last = x.rank
```

The demos:

```
"""
def rdiv0():
    rdivDemo([
        ["x1",0.34, 0.49, 0.51, 0.6],
        ["x2",6, 7, 8, 9] ])
"""
```

```
....
rank,    name,  med , iqr
-----
1,      x1,    51 , 11(* | ),0.34, 0.49, 0.51, 0.51, 0.60
2,      x2,    800 , 200( | ---- *---),6.00, 7.00, 8.00, 8.00, 9.00
....
```

```
"""
def rdiv1():
    rdivDemo([
        ["x1",0.1, 0.2, 0.3, 0.4],
        ["x2",0.1, 0.2, 0.3, 0.4],
        ["x3",6, 7, 8, 9] ])
"""
```

```
....
rank,    name,  med , iqr
-----
1,      x1,    30 , 20(* | ),0.10, 0.20, 0.30, 0.30, 0.40
2,      x2,    30 , 20(* | ),0.10, 0.20, 0.30, 0.30, 0.40
3,      x3,    800 , 200( | ---- *---),6.00, 7.00, 8.00, 8.00, 9.00
....
```

```
"""
def rdiv2():
    rdivDemo([
        ["x1",0.34, 0.49, 0.51, 0.6],
        ["x2",0.6, 0.7, 0.8, 0.9],
        ["x3",0.15, 0.25, 0.4, 0.35],
        ["x4",0.6, 0.7, 0.8, 0.9],
        ["x5",0.1, 0.2, 0.3, 0.4] ])
"""
```

```
....
rank,    name,  med , iqr
-----
1,      x5,    30 , 20(--- *--- | ),0.10, 0.20, 0.30, 0.30, 0.40
1,      x3,    35 , 15(--- *--- | ),0.15, 0.25, 0.35, 0.35, 0.40
2,      x1,    51 , 11(--- *--- | ),0.34, 0.49, 0.51, 0.51, 0.60
3,      x2,    80 , 20( | ---- *---),0.60, 0.70, 0.80, 0.80, 0.90
3,      x4,    80 , 20( | ---- *---),0.60, 0.70, 0.80, 0.80, 0.90
....
```

```
"""
def rdiv3():
    rdivDemo([
        ["x1",101, 100, 99, 101, 99.5],
        ["x2",101, 100, 99, 101, 100],
        ["x3",101, 100, 99.5, 101, 99],
        ["x4",101, 100, 99, 101, 100] ])
"""
```

```
....
rank,    name,  med , iqr
-----
1,      x1,  10000 , 150(----- *| ),99.00, 99.50, 100.00, 101.00, 101.00
1,      x2,  10000 , 100(----- *| ),99.00, 100.00, 100.00, 101.00, 101.00
1,      x3,  10000 , 150(----- *| ),99.00, 99.50, 100.00, 101.00, 101.00
1,      x4,  10000 , 100(----- *| ),99.00, 100.00, 100.00, 101.00, 101.00
....
```

```
"""
def rdiv4():
    rdivDemo([
        ["x1",11,12,13],
```


Sep 23, 14 14:00

csc710sbse: HW4:Theisen

Page 7/7

```

["x2",14,31,22],
["x3",23,24,31],
["x5",32,33,34]])
'''

....
rank, name, med, iqr
-----
1, x1, 1100, 0(*) | ),11.00, 11.00, 12.00, 13.00, 13.00
1, x2, 1400, 0( *| ),14.00, 14.00, 22.00, 31.00, 31.00
2, x3, 2300, 0( |* ),23.00, 23.00, 24.00, 31.00, 31.00
2, x5, 3200, 0( | * ),32.00, 32.00, 33.00, 34.00, 34.00
....

'''
def rdiv5():
    rdivDemo([
        ["x1",11,11,11],
        ["x2",11,11,11],
        ["x3",11,11,11]])
'''

....
rank, name, med, iqr
-----
1, x1, 1100, 0(*) | ),11.00, 11.00, 11.00, 11.00, 11.00
1, x2, 1100, 0(*) | ),11.00, 11.00, 11.00, 11.00, 11.00
1, x3, 1100, 0(*) | ),11.00, 11.00, 11.00, 11.00, 11.00
....

'''
def rdiv6():
    rdivDemo([
        ["x1",11,11,11],
        ["x2",11,11,11],
        ["x4",32,33,34,35]])
'''

....
rank, name, med, iqr
-----
1, x1, 1100, 0(*) | ),11.00, 11.00, 11.00, 11.00, 11.00
1, x2, 1100, 0(*) | ),11.00, 11.00, 11.00, 11.00, 11.00
2, x4, 3400, 200( | - * ),32.00, 33.00, 34.00, 34.00, 35.00
....

'''
def rdiv7():
    rdivDemo([
        ["x1"] + [rand()*0.5 for _ in range(256)],
        ["x2"] + [rand()*2 for _ in range(256)],
        ["x3"] + [rand() for _ in range(256)]
    ])
'''

....
rank, name, med, iqr
-----
1, x2, 25, 50( — * ————— ), 0.01, 0.09, 0.25, 0.47, 0.86
2, x3, 49, 47( ————— *| ————— ), 0.08, 0.29, 0.49, 0.66, 0.89
3, x1, 73, 37( —————|— * ——— ), 0.32, 0.57, 0.73, 0.86, 0.95
....

'''
def _rdivs():
    seed(1)
    rdiv0(); rdiv1(); rdiv2(); rdiv3();
    rdiv4(); rdiv5(); rdiv6(); rdiv7()

_rdivs()

```

Sep 23, 14 1:52

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from options import *

#Taken verbatim from the class website.

def pairs(lst):
    last=lst[0]
    for i in lst[1:]:
        yield last,i
        last = i

def xtile(lst, lo=0, hi=0.001, width=50,
         chops=[0.1, 0.3, 0.5, 0.7, 0.9],
         marks=["-", " ", " ", " ", "-"],
         bar="|", star="*", show="%3.0f"):
    """The function _xtile_ takes a list of (possibly)
    unsorted numbers and presents them as a horizontal
    xtile chart (in ascii format). The default is a
    contracted _quintile_ that shows the
    10,30,50,70,90 breaks in the data (but this can be
    changed- see the optional flags of the function).
    """
    def pos(p): return ordered[int((len(lst)*p))]
    def place(x):
        return int(width*float((x - lo))/(hi - lo))
    def pretty(lst):
        return ','.join([show % x for x in lst])
    ordered = sorted(lst)
    lo = min(lo, ordered[0])
    hi = max(hi, ordered[-1])
    what = [pos(p) for p in chops]
    where = [place(n) for n in what]
    out = [" "] * width
    for one, two in pairs(where):
        for i in range(one, two):
            out[i] = marks[0]
    marks = marks[1:]
    out[int(width/2)] = bar
    out[place(pos(0.5))] = star
    return ''.join(out) + " " + pretty(what)

```

Sep 23, 14 1:11

csc710sbse: HW4:Theisen

Page 1/1

```
from sim_anneal import *  
from max_walk_sat import *
```

Sep 23, 14 13:47

csc710sbse: HW4:Theisen

Page 1/1

```

#Structure from SA Lecture
import sys, random, math
sys.dont_write_bytecode = True

from options import *
from utils import *

myOpt = Options()

class MWS:
    debug = False
    name = "MaxWalkSat"

    def say(self, x):
        if self.debug:
            sys.stdout.write(str(x)); sys.stdout.flush()

    def specificRun(self, probability, klass):
        fon = klass
        XVarBest = fon.XVar
        eBest = e = 1
        k = 1
        temp = []
        self.say(int(math.fabs(eBest-1)*100))
        self.say(' ')
        for i in xrange(myOpt.mws_maxTries):
            fon.Chaos()
            for j in xrange(myOpt.mws_maxChanges):
                eNew = fon.Energy()
                if (eNew < myOpt.mws_threshold):
                    #% means found a solution and quit
                    self.say('%')
                    eBest = eNew
                    XVarBest = list(fon.XVar)
                    #print '\nQuitting...'
                    return eBest, XVarBest
            else:
                #modify random part of solution
                if probability > random.uniform(0,1):
                    fon.Neighbor()
                    self.say('+')
                #maximize for some random
            else:
                fon.BestNeighbor()
                self.say('.')
                temp.append(eNew)
                if (i+1)*(j+1) % 40 == 0 ^ len(temp) != 0:
                    #print ''
                    self.say(int(math.fabs(eNew-1)*100))
                    self.say(' ')
                    print xtile(temp,width=25,show=" %1.5f")
                    temp = []
                #print ''
            return -1, XVarBest

    def run(self, klass):
        theBest = -1
        valid = False
        eBest, XVarBest = self.specificRun(myOpt.mws_prob, klass)
        if eBest == -1:
            #print 'No Best Found for prob = ', i
            self.say('')
        else:
            theBest = eBest
            valid = True
        return theBest, valid

    def printOptions(self):
        print "MaxWalkSat Options:"
        print "Prob:", myOpt.mws_prob
        print "MaxTries:", myOpt.mws_maxTries, "MaxChanges", myOpt.mws_maxChanges
        print "Threshold:", myOpt.mws_threshold, "Slices:", myOpt.mws_slices

```

Sep 23, 14 13:34

csc710sbse: HW4:Theisen

Page 1/1

```

#Structure from SA Lecture
import sys, re, random, math
sys.dont_write_bytecode = True

from options import *
from utils import *

myOpt = Options()

class SA:
    name = "SA"

    def say(self, x):
        if myOpt.debug:
            sys.stdout.write(str(x)); sys.stdout.flush()

    def run(self, klass):
        sa = klass
        XVarBest = sa.XVar
        eBest = e = 1
        #print 'start energy: ', eBest
        k = 1
        temp = []
        self.say(int(math.fabs(eBest-1)*100))
        self.say(' ')
        while k < myOpt.sa_kmax:
            sa.Neighbor()
            eNew = sa.Energy()
            if eNew < eBest:
                eBest = eNew
                XVarBest = list(sa.XVar)
                self.say('!!')

            if eNew < e:
                e = eNew
                self.say('+')
            #Probability Check from SA Lecture
            elif math.exp(-1*(eNew-e)/(k/myOpt.sa_kmax*myOpt.sa_cooling)) < random.uniform(0,1):
                #P function should be between 0 and 1
                #more random hops early, then decreasing as time goes on
                sa.Chaos()
                self.say('?')
            self.say('.')
            k = k + 1
            temp.append(eBest)
            if k % 50 == 0 ^ k != myOpt.sa_kmax ^ len(temp) != 0:
                #print ''
                self.say(int(math.fabs(eBest-1)*100))
                self.say(' ')
                print xtile(temp,width=25,show=" %.15f")
                temp = []

        if myOpt.debug:
            #print '\nFound best - e: ', eBest
            #print 'Variables: '
            for vars in XVarBest:
                self.say(vars)
                self.say(",")
            #print "\n"
        return eBest, True

    def printOptions(self):
        print "SA Options:"
        print "KMAX:", myOpt.sa_kmax, "Cooling:", myOpt.sa_cooling

```

Sep 23, 14 1:46

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class ZDT1(Model):
    smin = 0
    smax = 1
    n = 30
    XVar = [random.uniform(smin, smax) for i in range (0, n)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        X = self.XVar
        f1 = X[0]
        g = 1+9*(np.sum([X[i] for i in range (1, self.n)]/(self.n-1))
        f2 = g*(1-np.sqrt(X[0]/g))
        return (math.fabs(f1-f2) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        X = self.XVar
        f1 = X[0]
        g = 1+9*(np.sum([X[i] for i in range (1, self.n)]/(self.n-1))
        f2 = g*(1-np.sqrt(X[0]/g))
        return math.fabs(f1-f2)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```

Sep 23, 14 13:09

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class ZDT3(Model):
    smin = 0
    smax = 1
    n = 30
    XVar = [random.uniform(smin, smax) for i in range (0, n)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        X = self.XVar
        f1 = X[0]
        g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
        f2 = g*(1-np.sqrt(X[0]/g)-(X[0]/g)*math.sin(10*math.pi*X[0]))
        return (math.fabs(f1-f2) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        X = self.XVar
        f1 = X[0]
        g = 1+9*(np.sum([X[i] for i in range (1, self.n)])/(self.n-1))
        f2 = g*(1-np.sqrt(X[0]/g)-(X[0]/g)*math.sin(10*math.pi*X[0]))
        return math.fabs(f1-f2)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```

Sep 23, 14 13:16

csc710sbse: HW4:Theisen

Page 1/1

```
from fonsaca_model import *  
from schaffer_model import *  
from kursawe_model import *  
from ZDT1_model import *  
from ZDT3_model import *  
from viennet3_model import *
```


Sep 23, 14 1:46

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class Fonseca(Model):
    n = 3
    smin = -4
    smax = 4
    XVar = [random.uniform(smin, smax) for i in range (0, 3)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        f1 = (1-math.e**(-np.sum([self.XVar[i]-(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
        f2 = (1-math.e**(-np.sum([self.XVar[i]+(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
        return (math.fabs(f1+f2) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        f1 = (1-math.exp(-np.sum([self.XVar[i]-(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
        f2 = (1-math.exp(-np.sum([self.XVar[i]+(1/np.sqrt(i+1))**2 for i in range (0, 3)])))
        return math.fabs(f1+f2)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```

Sep 23, 14 1:46

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

myOpt = Options()

class Kursawe(Model):
    n = 3
    smin = -5
    smax = 5
    XVar = [random.uniform(smin, smax) for i in range(0, 3)]
    XVarMax = XVar
    eMax = 0
    eMin = 0
    a = 0.8
    b = 3

    def Energy(self):
        X = self.XVar
        f1 = np.sum([-10*math.exp(-0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range(0, 3-1)])
        f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range(0, 3)])
        return (math.fabs(f1-f2) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        X = self.XVar
        f1 = np.sum([-10*math.exp(-0.2*(np.sqrt(X[i]**2+X[i]**2))) for i in range(0, 3-1)])
        f2 = np.sum([math.fabs(X[i])**self.a + 5*np.sin(X[i])**self.b for i in range(0, 3)])
        return math.fabs(f1-f2)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```

Sep 23, 14 1:48

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from options import *
myOpt = Options()
rand = random.random

class Model:
    #Default Values overwritten by subclass; should have better defaults, but...
    n = 1
    smin = 1
    smax = 1
    XVar = [random.uniform(smin, smax) for i in range(0, n)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        print "Energy Class Undefined!"

    def RawEnergy(self):
        print "RawEnergy Class Undefined!"

    def Neighbor(self):
        self.XVar[random.randint(0, self.n-1)] = random.uniform(self.smin, self.smax)

    def BestNeighbor(self):
        toChange = random.randint(0, self.n-1)
        toIncrement = (self.smax - self.smin) / myOpt.mws_slices
        curMax = 1
        maxVal = self.XVar[toChange]
        for i in xrange(myOpt.mws_slices):
            self.XVar[toChange] = self.smin + toIncrement
            x = self.Energy()
            if x < curMax:
                curMax = x
                maxVal = self.XVar[toChange]

    def Chaos(self):
        for vars in self.XVar:
            vars = random.uniform(self.smin, self.smax)

    def Baseline(self, numRuns):
        self.Chaos()
        self.eMax = self.eMin = self.RawEnergy()
        runs = 1
        while runs < numRuns:
            self.Neighbor()
            eNew = self.RawEnergy()
            if eNew > self.eMax: #find largest difference
                self.eMax = eNew
                self.XVarMax = self.XVar
                #print self.XVarMax, eNew
            if eNew < self.eMin: #find smallest difference
                self.eMin = eNew
                #print 'Min: ', self.XVar, eNew
            runs += 1
        #print 'Baseline: ', self.eMin, ' ', self.eMax

    def __init__(self):
        print "Default init Shouldn't be used!"

```

Sep 23, 14 1:46

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np

from model_base import *
from options import *

sys.dont_write_bytecode = True

class Schaffer(Model):
    n = 1
    smin = -10
    smax = 10
    XVar = [random.uniform(smin, smax) for i in range (0, 1)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        f1 = self.XVar[0]*self.XVar[0]
        f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
        return (math.fabs(f1+f2) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        f1 = self.XVar[0]*self.XVar[0]
        f2 = (self.XVar[0]-2)*(self.XVar[0]-2)
        return math.fabs(f1+f2)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```

Sep 23, 14 13:19

csc710sbse: HW4:Theisen

Page 1/1

```

#From Class Discussion 8/26/2014
from __future__ import division
import sys, re, random, math
import numpy as np
sys.dont_write_bytecode = True

from model_base import *
from options import *

class Viennet3(Model):
    smin = -3.0
    smax = 3
    n = 2
    XVar = [random.uniform(smin, smax) for i in range (0, n)]
    XVarMax = XVar
    eMax = 0
    eMin = 0

    def Energy(self):
        X = self.XVar
        f1 = 0.5*X[0]**2 + X[1]**2 + math.sin(X[0]**2+X[1]**2)
        f2 = (3*X[0]-2*X[1]+4)**2/8 + (X[0]-X[1]+1)**2/27 + 15
        f3 = 1/(X[0]+X[1]+1) - 1.1*math.e**(-X[0]**2-X[1]**2)
        return (math.fabs(f1-f2-f3) - self.eMin) / (self.eMax - self.eMin)

    def RawEnergy(self):
        X = self.XVar
        f1 = 0.5*X[0]**2 + X[1]**2 + math.sin(X[0]**2+X[1]**2)
        f2 = (3*X[0]-2*X[1]+4)**2/8 + (X[0]-X[1]+1)**2/27 + 15
        f3 = 1/(X[0]+X[1]+1) - 1.1*math.e**(-X[0]**2-X[1]**2)
        return math.fabs(f1-f2-f3)

    def __init__(self):
        self.Baseline(10000)
        self.XVar = self.XVarMax

```