

Oct 21, 14 13:15

"csc710sbse: hw6: Witschey"

Page 1/2

#### Schaffer										
rank	name	med	iqr			10%	30%	50%	70%	90%
5	1 DE,	2.0,	0.0	(*			2.00,	2.00,	2.00,	2.00
	2 GA,	2989552199.15,	5442183597.98	(-- * -----			93933299.54,	1588537631.53,	2989552199.15,	6613851395.02,
	2 SA,	4167583697.71,	7898876026.56	(- * --			140168713.91,	1219872808.62,	4240997459.91,	8681205262.52,
	2 MWS,	5059215093.07,	9882946504.86	(-- * -			137835411.19,	1568518268.39,	5059215093.07,	9829919731.48,
#### Kursawe										
rank	name	med	iqr			10%	30%	50%	70%	90%
10	1 DE,	-13.73,	0.0	(*			-13.73,	-13.73,	-13.73,	-13.73
	2 MWS,	-2.42,	4.34	(--- *--			-7.17,	-4.66,	-2.42,	-1.49,
	3 GA,	-1.4,	6.17	(--- *			-5.36,	-3.04,	-1.40,	-0.02,
	3 SA,	0.22,	7.5	(--- * -			-8.60,	-2.20,	0.22,	2.72,
#### Fonseca										
rank	name	med	iqr			10%	30%	50%	70%	90%
20	1 DE,	1.0,	0.0	(*			1.00,	1.00,	1.00,	1.00
	2 GA,	2.0,	0.02	(--- *			1.85,	1.98,	2.00,	2.00
	2 SA,	2.0,	0.08	(--- *			1.59,	1.98,	2.00,	2.00
	2 MWS,	2.0,	0.0	(--- *			2.00,	2.00,	2.00,	2.00
#### ZDT1										
rank	name	med	iqr			10%	30%	50%	70%	90%
25	1 DE,	1.81,	0.03	(*			1.78,	1.80,	1.81,	1.82,
	2 GA,	9.47,	1.04	(--- *			8.80,	9.13,	9.47,	9.85,
	2 SA,	9.72,	1.13	(--- *			8.80,	9.27,	9.74,	10.20,
	3 MWS,	14.51,	0.33	(--- *			14.18,	14.35,	14.51,	14.63,
#### ZDT3										
rank	name	med	iqr			10%	30%	50%	70%	90%
35	1 DE,	0.69,	0.01	(*			0.69,	0.69,	0.69,	0.70,
	2 GA,	4.15,	2.49	(--- *-----			1.82,	2.91,	4.15,	4.61,
	3 SA,	6.33,	3.94	(--- *			2.81,	4.58,	6.44,	7.83,
	4 MWS,	13.05,	6.67	(--- *)			6.59,	6.65,	13.05,	13.26,
#### Viennet3										
rank	name	med	iqr			10%	30%	50%	70%	90%
45	1 DE,	15.97,	0.0	(*			15.97,	15.97,	15.97,	15.97
	2 GA,	25.49,	10.63	(-- * ----			17.26,	21.05,	25.49,	28.67,
	2 SA,	25.65,	11.75	(-- * -			17.22,	21.51,	26.02,	30.65,
	3 MWS,	28.95,	9.08	(- * -			24.44,	25.75,	28.95,	34.05,
#### DTLZ7										
rank	name	med	iqr			10%	30%	50%	70%	90%
50	1 DE,	236.0,	9.04	(*			229.98,	233.35,	236.00,	240.25,
	2 SA,	4578.42,	3652.64	(*			2985.42,	3688.05,	4612.10,	6606.20,
	2 MWS,	5113.94,	449.23	(*			4716.99,	4782.82,	5113.94,	5196.45,
	2 GA,	8110.27,	9930.23	(*-			3178.46,	3485.22,	8110.27,	12628.67,
#### Schwefel(10)										
rank	name	med	iqr			10%	30%	50%	70%	90%
60	1 DE,	36239.93,	34935.61	(*			7977.19,	11559.85,	36239.93,	43811.26,
	2 GA,	567165.82,	293120.8	(--- * -			326757.90,	453731.48,	567165.82,	694528.84,
	2 MWS,	710795.46,	517251.2	(--- * --			248836.97,	286731.25,	710795.46,	788336.52,
	3 SA,	711333.43,	314585.05	(--- * -			458754.88,	612846.60,	712207.80,	839991.70,
#### Schwefel(20)										
rank	name	med	iqr			10%	30%	50%	70%	90%
65	1 DE,	727347.53,	153651.06	(*			582325.97,	626679.15,	727347.53,	763667.36,
	2 MWS,	2095384.33,	1126084.45	(--- * -			1334974.58,	1368757.08,	2095384.33,	2474456.60,
	2 GA,	2063333.41,	999203.02	(--- * -			1369614.06,	1740733.17,	2063333.41,	2569921.74,
	3 SA,	2804421.82,	1021144.26	(--- * -			1763026.75,	2268210.93,	2824707.03,	3062971.43,

#### Schwefel(40)												
rank	name	med	iqr				10%	30%	50%	70%	90%	
=====												
75	1	DE,	2620044.65,	1172436.43	(- *)	1765372.48,	2150225.34,	2620044.65,	3105723.70,	3494427.33	
	2	GA,	9956925.98,	2532069.74	(---)	7713756.49,	8884781.79,	9956925.98,	10939227.07,	12755142.88
	3	MWS,	10704016.93,	625403.08	()	10138226.45,	10212586.56,	10704016.93,	10799262.58,	10944175.73
	4	SA,	11293251.76,	2471764.19	(---)	8363810.18,	10288830.48,	11364866.24,	12271592.61,	14040412.02
80												

Oct 21, 14 0:51

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, print_function, unicode_literals

import random
import functools
5 import math
import itertools
import collections

10 def pretty_input(t):
    float_format = lambda x: '{:.2f}'.format(x)
    str_tuple = tuple(float_format(x) for x in t)
    return ', '.join(s for s in str_tuple)

15 def pairs(xs):
    # from https://docs.python.org/2/library/itertools.html
    a, b = itertools.tee(xs)
    next(b, None)
    for p in itertools.izip(a, b):
        yield p

class memo(object): # noqa -- TODO: rethink this name
25     '''adapted from https://github.com/timm/sbse14/wiki/basepy'''

    def __init__(self, **kwargs):
        self.__dict__.update(kwargs)

    def to_str(self, depth=0, indent=4, infix=' ', sep=', ', d=None):
        return '{' + self._to_str(
            depth=depth + 1,
            indent=indent,
            infix=infix,
            sep=sep,
35             d=self.__dict__ if d is None else d) + '}'

    def _to_str(self, depth, indent, infix, sep, d):
        after, before = [], []
        rv = ''
        for k in sorted([s for s in d.keys() if s[0] != '_']):
            val = d[k]
            if isinstance(val, memo) or type(val) == dict:
                after.append(k)
            else:
                before.append('{}{}{}'.format(k, infix, repr(val)))
        if before:
            rv += '\n' + ' ' * depth * indent
            rv += sep.join(before)
50         rv += '\n'

        for k in after:
            rv += ''.join([' ' * depth * indent, k, infix, '{}'])
            k = d[k]
            k = k if type(k) == dict else k.__dict__
            rv += ''.join([self._to_str(depth=depth+1, indent=indent,
                                     infix=infix, sep=sep, d=k),
                          ' ' * depth * indent,
                          '{}\n'])
60         return rv

    def memoize(f):
        'memoizer for single-arg functions'
        d = {}

        @functools.wraps(f)
        def wrapper(x):
70             try:
                return d[x]
            except KeyError:
                d[x] = f(x)

```

Oct 21, 14 0:51

"csc710sbse: hw6: Witschey"

Page 2/2

```

        return d[x]

75         return wrapper

    @memoize
    def memo_sqrt(x):
        return math.sqrt(x)

    def tuple_replace(t, replace_at, value):
85         return tuple(value if i == replace_at else v for i, v in enumerate(t))

    def random_index(x):
        if isinstance(x, dict):
            return random.choice(x.keys())
        if isinstance(x, collections.Iterable):
            return random.randint(0, len(x) - 1)
            raise ValueError('{} is not a dict or Iterable'.format(x))

95     class StringBuilder(object):
        def __init__(self, *args):
            self._s = ''.join(args)
            self._next = []

        def append(self, arg):
            'recurse through iterables in args, adding all strings to _next'
            'raises TypeError if it finds a non-Iterable non-string'
            if isinstance(arg, basestring):
                self._next.append(arg)
            elif isinstance(arg, collections.Iterable):
                map(self.append, arg)
            else:
                raise TypeError('{} not a string or iterable'.format(arg))

110        def __iadd__(self, arg):
            self.append(arg)
            return self

        def as_str(self):
            'build and cache _s if necessary, then return it.'
            if self._next:
                self._s += ''.join(self._next)
                self._next = []
            return self._s

        def __repr__(self):
            return "{}('{}')".format(self.__class__.__name__, self.as_str())

125     class NullObject(object):
        __slots__ = ()

        def __init__(self, *args, **kw):
            return None

        def __return_self(self, *name, **kw):
            return self

135        __getattr__ = __return_self
        __setattr__ = __return_self
        __iadd__ = __return_self
        __call__ = __return_self

        def __bool__(self, *args, **kw):
            return False
            __nonzero__ = __bool__

140

```

Oct 21, 14 3:36

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, print_function

import math

5 import base

def median(xs, is_sorted=False):
    # implementation from http://stackoverflow.com/a/10482734/3408454
10     if is_sorted:
        xs = sorted(xs)
        n = len(xs)
        return xs[n // 2] if n % 2 else (xs[n // 2] + xs[n // 2 - 1]) / 2

15     def mean(xs):
        return sum(xs) / len(xs)

20     def iqr(xs):
        n = len(xs)
        return xs[int(n * .75)] - xs[int(n * .25)]

25     def standard_deviation(xs, mean=None):
        if mean is None:
            mean = mean(xs)
            return math.sqrt((sum(x - mean for x in xs) ** 2)

30     def norm(x, lo, hi):
        return (x - lo) / (hi - lo)

35     def xtile(xs, lo=0, hi=0.001,
                width=50,
                chops=[0.1, 0.3, 0.5, 0.7, 0.9],
                marks=["-", " ", " ", "-", " "],
                bar="|", star="*",
40                 show="{: >6.2f}",
                as_list=False):
        """The function _xtile_ takes a list of (possibly) unsorted numbers and
        presents them as a horizontal xtile ascii chart. The default is a
        contracted _quintile_ that shows the 10,30,50,70,90 breaks in the data by
45         default. These breaks can be customized with the chops parameter.
        """

        xs = sorted(xs)

50         lo = min(lo, xs[0])
        hi = max(hi, xs[-1])
        if hi == lo:
            hi += .001 # ugh

55         out = [' '] * width

        pos = lambda p: xs[int(len(xs) * p)]
        place = lambda x: min(width-1, int(len(out) * norm(x, lo, hi)))

60         what = [pos(p) for p in chops]
        where = [place(n) for n in what]

        for one, two in base.pairs(where):
            for i in range(one, two):
65                 out[i] = marks[0]
            marks = marks[1:]

        out[int(width / 2)] = bar
        out[place(pos(0.5))] = star

70         if as_list:
            rv = ['(' + ''.join(out) + ")"]
            rv.extend(show % x for x in what)

```

Oct 21, 14 3:36

"csc710sbse: hw6: Witschey"

Page 2/2

```

        return rv

75         return ''.join(out) + "," + ' '.join([show.format(x) for x in what])

```

Oct 21, 14 3:47

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, print_function

import random
import functools
5 import collections
import itertools

from sortedcontainers import SortedList

10 from witschey import base

class Log(object):
    """Keep a random sample of stuff seen so far. Based on Dr. Menzies'
    implementation."""
15
    MAX_SIZE = 256

    def __init__(self, inits=None, label=None, max_size=MAX_SIZE):
20
        self._cache = SortedList()
        self._report = None
        self.label = label or ''
        self._n = 0
        self.max_size = max_size
25
        self._valid_statistics = False
        self._invalidate_statistics()
        if inits:
            map(self.__iadd__, inits)

30
    def random_index(self):
        return base.random_index(self._cache)

    @classmethod
    def wrap(cls, x, max_size=MAX_SIZE):
35
        if isinstance(x, cls):
            return x
        return cls(inits=x, max_size=max_size)

    def __len__(self):
40
        return len(self._cache)

    def extend(self, xs):
        if not isinstance(xs, collections.Iterable):
            raise TypeError()
45
        map(self.__iadd__, xs)

    def __iadd__(self, x):
        if x is None:
            return x
50
        self._n += 1

        if isinstance(x, Log):
            map(self.__iadd__, x._cache)
55
        changed = False

        # if cache has room, add item
        if self.max_size is None or len(self._cache) < self.max_size:
60
            changed = True
            self._cache.add(x)
        # cache is full: maybe replace an old item
        else:
            # items less likely to be replaced later in the run:
            # leads to uniform sample of entire run
65
            if random.random() <= self.max_size / len(self):
                changed = True
                self._cache.remove(random.choice(self._cache))
                self._cache.add(x)
70

        if changed:
            self._invalidate_statistics()
            self._change(x)

```

Oct 21, 14 3:47

"csc710sbse: hw6: Witschey"

Page 2/2

```

75
        return self

    def __add__(self, x, max_size=MAX_SIZE):
        inits = itertools.chain(self._cache, x._cache)
        return self.__class__(inits=inits, max_size=max_size)
80

    def any(self):
        return random.choice(self._cache)

    def report(self):
85
        if self._report is None:
            self._report = self._generate_report()
        return self._report

    def setup(self):
90
        raise NotImplementedError()

    def as_list(self):
        return self._cache.as_list()

95
    def _invalidate_statistics(self):
        """
        default implementation. if _valid_statistics is something other than
        a boolean, reimplement!
        """
100
        self._valid_statistics = False

    def ish(self, *args, **kwargs):
        raise NotImplementedError()

105
    def _change(self, x):
        """
        override to add incremental updating functionality
        pass
110

    def _prepare_data(self):
        s = '_prepare_data() not implemented for ' + self.__class__.__name__
        raise NotImplementedError(s)

115
    def contents(self):
        return self._cache.as_list()

    def statistic(f):
120
        """
        decorator for log functions that return statistics about contents.
        if _valid_statistics is False, generate valid stats before calling
        the wrapped function.
        """
125
        @functools.wraps(f)
        def wrapper(*args, **kwargs):
            self = args[0]
            if not self._valid_statistics:
                self._prepare_data()
130
            return f(*args, **kwargs)

        return wrapper

```

Oct 18, 14 23:57

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division

from log import Log
from witschey import base
5 from witschey import basic_stats

class NumberLog(Log):
10     def __init__(self, *args, **kwargs):
        super(NumberLog, self).__init__(*args, **kwargs)
        self._invalidate_statistics()

15     @property
        def hi(self):
            return self._cache[-1] # assumes SortedList implementation

    @property
20     def lo(self):
        return self._cache[0] # assumes SortedList implementation

    def _invalidate_statistics(self):
        self._cached_mean, self._cached_median = None, None
25         self._cached_sd, self._cached_iqr = None, None

        super(NumberLog, self)._invalidate_statistics()

    def norm(self, x):
30         "normalize the argument with respect to maximum and minimum"
        if self.hi == self.lo:
            raise ValueError('hi and lo of {} are equal'.format(self.__name__))
        return basic_stats.norm(x, self.lo, self.hi)

35     def _prepare_data(self):
        if not self._valid_statistics:
            pass
        self._valid_statistics = True

40     def _generate_report(self):
        return base.memo(median=self.median(), iqr=self.iqr(),
                        lo=self.lo, hi=self.hi)

    def ish(self, f=0.1):
45         """return a num likely to be similar to/representative of
        nums in the distribution"""
        return self.any() + f*(self.any() - self.any())

    def median(self):
50         if self._cached_median is not None:
            return self._cached_median
        self._cached_median = basic_stats.median(self._cache)
        return self._cached_median

55     def mean(self):
        if self._cached_mean is not None:
            return self._cached_mean
        self._cached_mean = basic_stats.mean(self._cache)
        return self._cached_mean

60     def standard_deviation(self):
        if self._cached_sd is not None:
            return self._cached_sd
        self._cached_sd = basic_stats.standard_deviation(
65             self._cache, mean=self.mean())
        return self._cached_sd

    def iqr(self):
70         if self._cached_iqr is not None:
            return self._cached_iqr
        self._cached_iqr = basic_stats.iqr(self._cache)
        return self._cached_iqr

```

Oct 18, 14 23:57

"csc710sbse: hw6: Witschey"

Page 2/2

```

75     def xtile(self, *args, **kw):
        return basic_stats.xtile(self._cache, *args, **kw)

    def better(self, log2):
        if log2 is None:
            return ValueError
80         if not self._cache or not log2._cache:
            return False
        if self.median() < log2.median():
            return True
        if self.iqr() < log2.iqr():
85         return True
        return False

```

Oct 15, 14 16:45

"csc710sbse: hw6: Witschey"

Page 1/1

```

from model import Model
from independent_variable import IndependentVariable
from schaffer import Schaffer
from kursawe import Kursawe
5 from fonseca import Fonseca
from zdt1 import ZDT1
from zdt3 import ZDT3
from viennet3 import Viennet3
from dtlz7 import DTLZ7
10 from schwefel import Schwefel

__all__ = [Model, IndependentVariable, Schaffer, Kursawe, Fonseca,
           ZDT1, ZDT3, Viennet3, DTLZ7, Schwefel]
```

Oct 18, 14 18:59

"csc710sbse: hw6: Witschey"

Page 1/1

```

from __future__ import division, print_function

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py
5
from abc import ABCMeta
from collections import namedtuple

10 ModelIO = namedtuple('ModelIO', ('xs', 'ys', 'energy'))

class Model(object):
    # allows us to get all subclasses with __subclasses__()
15    __metaclass__ = ABCMeta

    def __init__(self, independents=None, dependents=None,
                 energy_min=None, energy_max=None,
                 enforce_energy_constraints=False):
20        if independents is None or dependents is None:
            raise ValueError

        self.xs = independents
        self.ys = dependents
25        self.energy_max = energy_max
        self.energy_min = energy_min
        self.enforce_energy_constraints = enforce_energy_constraints

    def normalize(self, x):
30        n = x - self.energy_min
        d = self.energy_max - self.energy_min
        try:
            return n / d
        except ZeroDivisionError:
35            return 0.5

    def random_input_vector(self):
        return tuple(x() for x in self.xs)

40    def __call__(self, xs, io=False):
        ys = tuple(y(xs) for y in self.ys)
        energy = sum(ys)

        if self.enforce_energy_constraints:
45            energy_errmsg = 'current energy {} not in range [{}, {}]'
            energy_errmsg = energy_errmsg.format(
                energy, self.energy_min, self.energy_max)

            if self.energy_min is None or self.energy_min > energy:
                if self.enforce_energy_constraints:
                    raise ValueError(energy_errmsg)
50                self.energy_min = energy

            if self.energy_max is None or energy > self.energy_max:
                if self.enforce_energy_constraints:
                    raise ValueError(energy_errmsg)
55                self.energy_max = energy

        if io:
            return ModelIO(xs, ys, energy)
60
        return ys

    def energy(self, ys, norm=False):
        rv = sum(ys)
65        return self.normalize(rv) if norm else rv

    def compute_model_io(self, xs):
        ys = self(xs)
        return ModelIO(xs, ys, self.energy(ys))
70

    def random_model_io(self):
        return self.compute_model_io(self.random_input_vector())
```

Oct 19, 14 0:45

"csc710sbse: hw6: Witschey"

Page 1/1

```

from __future__ import division, print_function
# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

5 import random

class IndependentVariable(object):
    def __init__(self, lo=None, hi=None, type=float):
10         self.lo = lo
            self.hi = hi
            self.type = type

    def __call__(self):
15         if self.type == float:
            f = random.uniform
        elif self.type == int:
            f = random.randint

20         return f(self.lo, self.hi)

    def clip(self, x):
        return max(self.lo, min(self.hi, x))

```

Oct 15, 14 20:49

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noga

10 class DTLZ7(Model):
    def __init__(self, ivs=30, dvs=20):

        # dynamically generate these suckers
        # h/t http://stackoverflow.com/a/13184536/3408454
15         generated_fs = []
        for x in xrange(1, dvs):
            f = lambda xs: xs[x]
            f.__name__ = 'f{}'.format(x)
            generated_fs.append(f)

20         def g(xs):
            # avoid divide by 0 errors
            denom = abs(xs[-1]) or .0001
            return 1 + (9 / denom) * sum(xs)

25         def h(xs, fs=generated_fs, g=g):
            s = 0
            for f in fs:
30                 fxs = f(xs)
                    a = fxs / (1 + g(xs))
                    b = 1 + math.sin(3 * math.pi * fxs)
                    s += a * b

            return dvs - s

35         def final_f(xs):
            return (1 + g(xs)) * h(xs)
            final_f.__name__ = 'f{}'.format(dvs)

40         fs = tuple(generated_fs + [final_f])

        independents = tuple(IV(lo=0, hi=1) for _ in xrange(ivs))
        super(DTLZ7, self).__init__(independents=independents, dependents=fs)

```


Oct 15, 14 16:54

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noqa
from witschey.base import memo_sqrt
10

class Fonseca(Model):
    def __init__(self, ivs=3):
        ivs = tuple(IV(lo=-4, hi=4) for _ in xrange(ivs - 1))
15
        def f1(xs):
            e = sum((x - (1 / memo_sqrt(i+1))) ** 2 for i, x in enumerate(xs))
            return 1 - math.exp(-e)

        def f2(xs):
20            e = sum((x + (1 / memo_sqrt(i+1))) ** 2 for i, x in enumerate(xs))
            return 1 - math.exp(-e)

        super(Fonseca, self).__init__(independents=ivs, dependents=(f1, f2))

```

Oct 15, 14 20:18

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noqa
10

class Kursawe(Model):
    def __init__(self, ivs=3, a=0.8, b=3):
        ivs = tuple(IV(lo=-5, hi=5) for _ in xrange(ivs - 1))
        self.a = a
        self.b = b
15
        def f1(xs):
            rv = 0
            for i in xrange(len(xs) - 1):
                exponent = (-0.2) * math.sqrt(xs[i] ** 2 + xs[i+1] ** 2)
                rv += -10 * math.exp(exponent)
            return rv
20
        def f2(xs):
            f = lambda x: (math.fabs(x)**self.a) + (5 * math.sin(x)**self.b)
            return sum(f(x) for x in xs)
25
        super(Kursawe, self).__init__(independents=ivs, dependents=(f1, f2))

```

Oct 21, 14 2:56

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from model import Model
5 from independent_variable import IndependentVariable as IV # noqa

class Schaffer(Model):

10     def __init__(self, ivs=1):
        independents = tuple(IV(lo=-10 ** 5, hi=10 ** 5) for _ in xrange(ivs))

        # use def instead of lambdas so the functions keep their __name__s
        def f1(xs):
15             return sum(x ** 2 for x in xs)

        def f2(xs):
            return sum((x - 2) ** 2 for x in xs)

20     super(Schaffer, self).__init__(
        independents=independents, dependents=(f1, f2))

```

Oct 15, 14 20:22

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noqa

10 class Viennet3(Model):

    def __init__(self):

15         def f1(xs):
            x_1sq = xs[0] ** 2
            x_2sq = xs[1] ** 2
            a = 0.5 * x_1sq
            b = math.sin(x_1sq + x_2sq)
20             return a + x_2sq + b

        def f2(xs):
            x_1 = xs[0]
            x_2 = xs[1]

25             a = ((3 * x_1 - 2 * x_2 + 4) ** 2) / 8
            b = ((x_1 + x_2 + 1) ** 2) / 27

            return a + b + 15

30         def f3(xs):
            x_1sq = xs[0] ** 2
            x_2sq = xs[1] ** 2

            a = 1 / (x_1sq + x_2sq + 1)
35             b = 1.1 * math.exp(-x_1sq - x_2sq)

            return a - b

40         ivs = (IV(lo=-3, hi=3), IV(lo=-3, hi=3))

        super(Viennet3, self).__init__(
            independents=ivs, dependents=(f1, f2, f3))

```

Oct 15, 14 20:18

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noga

10 class ZDT1(Model):
    def __init__(self, ivs=30):

        def g(xs):
15             return 1 + 9 * sum(xs[1:]) / (len(xs) - 1)

        def f1(xs):
            return xs[0]

20        def f2(xs):
            gxs = g(xs)
            return gxs * (1 - math.sqrt(xs[0] / gxs))

        ivs = tuple(IV(lo=0, hi=1) for _ in xrange(30))
25        super(ZDT1, self).__init__(independents=ivs, dependents=(f1, f2, g))

```

Oct 15, 14 20:28

"csc710sbse: hw6: Witschey"

Page 1/1

```

# all adapted from Dr. Tim Menzies' model code:
# https://github.com/timm/sbse14/blob/master/models.py

from __future__ import division
5 import math

from model import Model
from independent_variable import IndependentVariable as IV # noga
10 from witschey.base import memo_sqrt

class ZDT3(Model):

15     def __init__(self, ivs=30):

        def g(xs):
            return 1 + 9 * sum(xs[1:]) / (len(xs) - 1)

20        def f1(xs):
            return xs[0]

        def f2(xs):
            gxs = g(xs)
25            a = 1 - memo_sqrt(xs[0] / gxs) - (xs[0] / gxs)
            a *= math.sin(10 * math.pi * xs[0])
            return gxs * a

        ivs = tuple(IV(lo=0, hi=1) for _ in xrange(30))
30        super(ZDT3, self).__init__(independents=ivs, dependents=(f1, f2, g))

```

Oct 21, 14 12:50

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, unicode_literals

from datetime import datetime
import abc
5 from types import NoneType
from collections import namedtuple, OrderedDict

from witschey.base import memo
from witschey.models import Model
10 from witschey.config import CONFIG

class Searcher(object):
    # allows us to get all subclasses with __subclasses__()
15     __metaclass__ = abc.ABCMeta

    def __new__(cls, *args, **kwargs):
        # construct our object
        future_self = super(Searcher, cls).__new__(cls, *args, **kwargs)
20
        name = cls.__name__
        # initialize a dict with searcher's name
        # and the initialization time
        d = dict(searcher=name, initialized=datetime.now())
25
        # if there are global options for this class or its bases in CONFIG
        for k in [name] + [k.__name__ for k in cls.__bases__]:
            if hasattr(CONFIG, k):
                # add them to the dict
30                d.update(getattr(CONFIG, k).__dict__)

        # then, add the kwargs to the constructor call to the dict.
        # NB: this happens after adding options from The, so
        # call-specific options override the globals
35        d.update(kwargs)

        # set our spec with the contents of the dict
        future_self.spec = memo(**d)
40
        return future_self

    def __init__(self, model, *args, **kw):
        self.model = model()

45    def run(*args, **kwargs):
        raise NotImplementedError()

class SearcherConfig(object):
50
    def __init__(self, searcher=None, model=None, **kwargs):
        self.searcher, self.model = searcher, model
        self._kw_dict = kwargs

55    def get_searcher(self, searcher=None, model=None, **kwargs):
        s = searcher or self.searcher
        m = model or self.model
        kw = self._kw_dict.copy().update(kwargs) or {}
        return s(m, **kw)
60
    @property
    def searcher(self):
        return self._searcher

65    @searcher.setter
    def searcher(self, value):
        if isinstance(value, NoneType) or isinstance(value, Searcher):
            self._searcher = value
        else:
70            raise TypeError('{} is not a Searcher or None'.format(value))

    @property
    def model(self):

```

Oct 21, 14 12:50

"csc710sbse: hw6: Witschey"

Page 2/2

```

        return self._model

75
    @model.setter
    def model(self, value):
        if isinstance(value, NoneType) or isinstance(value, Model):
            self._model = value
80        else:
            raise TypeError('{} is not a Model or None'.format(value))

    def update(self, searcher=None, model=None, **kwargs):
        if searcher is not None:
85            self.searcher = searcher
        if model is not None:
            self.model = model
            self._kw_dict.update(kwargs)

90    def as_dict(self):
        "returns a OrderedDict with the searcher and model first"
        return OrderedDict(searcher=self._searcher,
                           model=self._model, **self._kw_dict)

95    def __repr__(self):
        kw_string = ', '.join('{}={}'.format(k, v)
                               for k, v in self.as_dict().iteritems())
        return '{}({})'.format(self.__class__.__name__, kw_string)

100 SearchReport = namedtuple('SearchReport',
                           ['best', 'best_era', 'evaluations', 'searcher',
                            'spec'])

```

Oct 21, 14 12:50

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, print_function

import random
import math

5  from searcher import Searcher, SearchReport
    from witschey.base import NullObject, StringBuilder
    from witschey.log import NumberLog

10  class SimulatedAnnealer(Searcher):
        def __init__(self, model, *args, **kw):
            super(SimulatedAnnealer, self).__init__(model=model, *args, **kw)

15  def _get_neighbor(self, model_io):
        n_gen = (model_io.xs[i]
                 if random.random() < self.spec.p_mutation else v
                 for i, v in enumerate(self.model.random_input_vector()))
        return self.model(tuple(n_gen), io=True)

20  def run(self, text_report=True):
        report = StringBuilder() if text_report else NullObject()
        current = self.model.random_model_io()
        best = current # assumes current is immutable
        self.lives = 4
        current_era_energies = NumberLog(max_size=None)
        best_era = None
        evals = None

30  for k in range(self.spec.iterations):
        if self.lives <= 0 and self.spec.terminate_early:
            evals = k
            break
        prev_era_energies = current_era_energies

35  neighbor = self._get_neighbor(current)
        current_era_energies += neighbor.energy

        if neighbor.energy < best.energy:
            best, current = neighbor, neighbor
            report += '!'

        if neighbor.energy < current.energy:
            current = neighbor
            report += '+'
45  else:
            cnorm = self.model.normalize(current.energy)
            nnorm = self.model.normalize(neighbor.energy)
            temp = k / self.spec.iterations
            if self._good_idea(cnorm, nnorm, temp) < random.random():
                current = neighbor
                report += '?'

        report += '.'

55  if k % self.spec.era_length == 0 and k != 0:
            report += ('\n', '{:.2}'.format(best.energy), ' ')
            if not best_era:
                best_era = current_era_energies

60  try:
            improved = current_era_energies.better(prev_era_energies)
        except ValueError:
            improved = False
        if improved:
            best_era = current_era_energies
        else:
            self.lives -= 1
            current_era_energies = NumberLog()

70  if evals is None:
            evals = self.spec.iterations
        rv = SearchReport(best=best.energy, evaluations=evals,

```

Oct 21, 14 12:50

"csc710sbse: hw6: Witschey"

Page 2/2

```

        best_era=best_era, spec=self.spec,
        searcher=self.__class__)

75  return rv

    def _good_idea(self, old, new, temp):
        """
80  sets the threshold we compare to to decide whether to jump

        returns e^-((new-old)/temp)
        """
        numerator = new - old

85  if not 0 <= numerator <= 1:
            numerator = old - new
        try:
            exponent = numerator / temp
90  except ZeroDivisionError:
            return 0
        rv = math.exp(-exponent)
        if rv > 1:
            raise ValueError('p returning greater than one',
95  rv, old, new, temp)
        return rv * self.spec.cooling_factor

```

Oct 21, 14 12:52

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division

import random
import numpy as np

5  from searcher import Searcher, SearchReport
    from witschey import base
    from witschey.base import tuple_replace, StringBuilder, NullObject
    from witschey.log import NumberLog

10  class MaxWalkSat(Searcher):

    def __init__(self, model, *args, **kw):
15         super(MaxWalkSat, self).__init__(model=model, *args, **kw)

    def _local_search_xs(self, bottom, top, n=10):
        '''divide the space from bottom to top into n partitions, then
        randomly sample within each partition'''
20         chunk_length = (top - bottom) / n

        for a in np.arange(bottom, top, chunk_length):
            yield random.uniform(a, a + chunk_length)

25     def _update(self, improvement_char, dimension=None, value=None):
        '''calculate the next value from the model and update state as
        necessary'''
        # check for invalid input
        if value is not None and dimension is None:
30             err = 'cannot call _update with specified value but no dimension'
            raise ValueError(err)

        if dimension is None:
            dimension = base.random_index(self._current.xs)
35         if value is None:
            # get random value if no value input
            value = self.model.xs[dimension]()

        # generate and evaluate input vector
40         new_xs = tuple_replace(self._current.xs, dimension, value)
        self._current = self.model(new_xs, io=True)
        self._evals += 1
        self._current_era += self._current.energy

45         # compare to previous best and update as necessary
        if self._current.energy < self._best.energy:
            self._best = self._current
            self._report += improvement_char
        else:
50             self._report += '.'

    def run(self, text_report=True):
        '''run MaxWalkSat on self.model'''

55         # current ModelIO to evaluate and mutate
        self._current = self.model.random_model_io()
        self._best = self._current
        # initialize and update log variables to track values by era
        self._current_era = NumberLog()
        self._current_era += self._current.energy
60         best_era = None
        # bookkeeping variables
        self._evals = 0
        lives = 4
65         self._report = StringBuilder() if text_report else NullObject()
        terminate = False

        while self._evals < self.spec.iterations and not terminate:
            # get the generator for a random independent variable
70             if self.spec.p_mutation > random.random():
                # if not searching a dimension, mutate randomly
                self._update('++')

```

Oct 21, 14 12:52

"csc710sbse: hw6: Witschey"

Page 2/2

```

else:
75     # if doing a local search, choose a dimension
        dimension = base.random_index(self._current.xs)
        search_iv = self.model.xs[dimension]
        # make sure local search ends at era end
        max_search = self.spec.era_length - (self._evals
80             % self.spec.era_length)

        n = min(10, max_search)
        # then try points all along the dimension
        lo, hi = search_iv.lo, search_iv.hi
        for j in self._local_search_xs(lo, hi, n):
85             self._update('|', dimension=dimension, value=j)

        # end-of-era bookkeeping
        if self._evals % self.spec.era_length == 0:
            self._report += ('\n{: .2}'.format(self._best.energy), ' ')
90             # _prev_era won't exist in era 0, so account for that case
            try:
                improved = self._current_era.better(self._prev_era)
            except AttributeError:
                improved = False
95             self._prev_era = self._current_era

        # track best_era
        if improved or best_era is None:
            best_era = self._current_era
100         else:
            lives -= 1

        if lives <= 0:
            terminate = True
        else:
            self._current_era = NumberLog()

110         return SearchReport(best=self._best.energy,
                               best_era=best_era,
                               evaluations=self._evals,
                               searcher=self.__class__,
                               spec=self.spec)

```

Oct 21, 14 12:04

"csc710sbse: hw6: Witschey"

Page 1/2

```

from __future__ import division, print_function

import itertools
import random

5
from witschey import base
from searcher import Searcher, SearchReport
from witschey.log import NumberLog

10
# adapted from Chris Theisen's code
# his code provided the shell that I worked in and styled to my liking
# Structure from:
# www.cleveralgorithms.com/nature-inspired/evolution/genetic_algorithm.html

15
class GeneticAlgorithm(Searcher):

    def __init__(self, model, *args, **kw):
        super(GeneticAlgorithm, self).__init__(model=model, *args, **kw)

20
    def mutate(self, child):
        i = base.random_index(child)
        return base.tuple_replace(child, i, self.model.xs[i]())

25
    def crossover(self, parent1, parent2, xovers=1):
        if len(parent1) != len(parent2):
            raise ValueError('parents must be same length to breed')
        if len(parent1) == 1:
            return random.choice((parent1, parent2))

30
        if xovers < 1:
            raise ValueError('cannot have fewer than 1 crossover')
        xovers = min(len(parent1) - 2, xovers)
        xovers = sorted(random.sample(xrange(1, len(parent1) - 1), xovers))
        x_pts = itertools.chain((0,), xovers, (None,))

35
        cycle_parents = itertools.cycle((parent1, parent2))
        parent_point_zip = itertools.izip(cycle_parents, base.pairs(x_pts))

40
        segments = [itertools.islice(parent, p[0], p[1])
                     for parent, p in parent_point_zip]

        return tuple(itertools.chain(*segments))

45
    def select_parents(self, population, output_size):
        """generates all possible parent pairs from population, clipped to
        output_size
        """
        fore = itertools.combinations(population, 2)
        back = itertools.combinations(reversed(population), 2)
        all_parents = set(fore).union(set(back))
        if len(all_parents) < output_size:
            return all_parents
        return random.sample(all_parents, output_size)

55
    def run(self, text_report=True):
        rand_vect = lambda: self.model.random_input_vector()
        pop_size = self.spec.population_size
        init_xs = tuple(rand_vect() for _ in xrange(pop_size))
        energy = lambda x: x.energy
        best_era = None

        report = base.StringBuilder() if text_report else base.NullObject()

65
        population = tuple(self.model.compute_model_io(xs) for xs in init_xs)

        best = min(population, key=energy)

        evals, lives = 0, 4

70
        for gen in xrange(self.spec.iterations):
            if evals > self.spec.iterations or lives <= 0:
                break

```

Oct 21, 14 12:04

"csc710sbse: hw6: Witschey"

Page 2/2

```

75
        children = []
        for parent1, parent2 in self.select_parents(population, pop_size):
            xs = self.crossover(parent1.xs, parent2.xs, 2)
            if random.random() < self.spec.p_mutation:
                self.mutate(xs)
80
            child = self.model(xs, io=True)
            children.append(child)

        best_in_pop = min(children, key=energy)

85
        prev_best_energy = best.energy
        best = min(best, best_in_pop, key=energy)

        report += str(best.energy)
        report += ('+' if x.energy < prev_best_energy else '.')
90
        for x in children)
        report += '\n'

        population = children
        evals += len(population)

95
        energies = NumberLog(inits=(c.energy for c in children))
        try:
            improved = energies.better(prev_energies)
        except NameError:
            improved = False
100
        prev_energies = energies # noqa: flake8 doesn't catch use above

        if improved:
            best_era = energies
105
        else:
            lives -= 1

        if best_era is None:
            best_era = energies

110
        return SearchReport(best=best.energy,
                             best_era=best_era,
                             evaluations=evals,
                             searcher=self.__class__,
115
                             spec=self.spec)

```

Oct 21, 14 11:59

"csc710sbse: hw6: Witschey"

Page 1/5

```

from __future__ import division
import sys
import random
import math

5
import texttable
from basic_stats import xtile, median
from witschey import base

10 # flake8: noqa

"""

### Standard Accumulator for Numbers

15 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.

20 """
class Num:
    "An Accumulator for numbers"
    def __init__(i,name,inits=[]):
        i.n = i.m2 = i.mu = 0.0
        i.all=[]
        i._median=None
        i.name = name
        i.rank = 0
        for x in inits: 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):
        i.all = sorted(i.all)
        n = int(len(i.all)*0.25)
        return i.all[n] , i.all[n * 2] , i.all[n * 3]
    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)]

    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 and 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:

```

Oct 21, 14 11:59

"csc710sbse: hw6: Witschey"

Page 2/5

```

        if h3 and 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
80 #-----
    lst1 = sorted(lst1, reverse=True)
    lst2 = sorted(lst2, reverse=True)
    n1 = len(lst1)
    n2 = len(lst2)
85 t1 = base.memo(l=lst1,j=0,eq=0,gt=0,n=n1)
    t2 = base.memo(l=lst2,j=0,eq=0,gt=0,n=n2)
    gt,eq= loop(t1, t1, t2)
    return gt/(n1*n2) + eq/2/(n1*n2)

90
"""### Non-Parametric Hypothesis Testing

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

[efron01]: http://goo.gl/14n8Wf "Bradley Efron and R.J. Tibshirani. An Introduct
ion to the Bootstrap (Chapman & Hall/CRC Monographs on Statistics & Applied Prob
ability), 1993"

100
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.

105 """
def sampleWithReplacement(lst):
    "returns a list same size as list"
    def any(n) : return random.uniform(0,n)
    def one(lst): return lst[ int(any(len(lst))) ]
    return [one(lst) for _ in lst]
"""

115 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.

120 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.

125 """
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
"""

140 The rest is just details:

+ Efron advises
to make the mean of the populations the same (see
the _yhat,zhat_ stuff shown below).

```


Oct 21, 14 11:59

"csc710sbse: hw6: Witschey"

Page 3/5

```

145 + 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):
150 """The bootstrap hypothesis test from
    p220 to 223 of Efron's book 'An
    introduction to the bootstrap.'"""
    class total():
        "quick and dirty data collector"
        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
160     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]
165    zhat = [z1 - z.mu + x.mu for z1 in z.all]
    bigger = 0.0
    for i in range(b):
        if testStatistic(total(sampleWithReplacement(yhat)),
170            total(sampleWithReplacement(zhat))) > tobs:
            bigger += 1
    return bigger / b < conf

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

"""

180 ## Saner Hypothesis Testing

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

185 + 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
190     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;
195
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  $\log_2 16 = 4$ 
hypothesis tests and confidence  $\geq 0.99$ .

200 + But if did this with the 120 all-pairs comparisons of the 16 treatments, we would
    have total confidence  $\geq 0.99$ .

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

"""
205 def scottknott(data,cohen=0.3,small=3,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)
210    same = lambda l, r: not different(l.all,r.all)
    big = lambda n: n > small

```

Oct 21, 14 11:59

"csc710sbse: hw6: Witschey"

Page 4/5

```

        return rdiv(data,all,minMu,big,same,epsilon)

215 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
220     """Looks for ways to split sorted data,
    Recurses into each split. Assigns a 'rank' number
    to all the leaf splits found in this way.
    """
225     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:
230             # 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
235             for part in parts:
                part.rank = rank
            return rank
        recurse(sorted(data),all)
        return data

240     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
245     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.
        """
255         cut,left,right = None,None,None
        before, mu = 0, all.mu
        for i,l,r in leftRight(parts,epsilon):
            if big(l.n) and big(r.n):
                n = all.n * 1.0
                now = l.n/n*(mu- l.mu)**2 + r.n/n*(mu- r.mu)**2
260                 if now > before:
                    before,cut,left,right = now,i,l,r
        return cut,left,right

    def leftRight(parts,epsilon=0.01):
265         """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
        """
270         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:
280                 if parts[i]._median - parts[i-1]._median > epsilon:
                    yield i,left,rights[i]
                    left += one
        """
285     ## Putting it All Together

```

Driver for the demos:

```

"""
290 def rdiv_report(data):
    rows = []
    def z(x):
        return int(100 * (x - lo) / (hi - lo + 0.00001))
    data = map(lambda lst: Num(lst[0], lst[1:]),
295         data)
    ranks = []
    for x in scottknott(data):
        ranks += [(x.rank, x.median(), x)]
    all = []
300 for _, __, x in sorted(ranks): all += x.all
    all = sorted(all)
    lo, hi = all[0], all[-1]
    last = None
    rows.append(['rank', 'name', 'med', 'iqr', '',
305         '10%', '30%', '50%', '70%', '90%'])
    for _, __, x in sorted(ranks):
        q1, q2, q3 = (round(q, 2) for q in x.quartiles())
        xtile_out = xtile(x.all, lo=lo, hi=hi, width=30, show="%5.2f", as_list=True)
        row_xtile = [xtile_out[0]] + map(lambda x: x + ', ', xtile_out[1:-1]) + \
310         [xtile_out[-1]]
        rows.append([x.rank+1] +
            map(lambda y: str(y) + ', ', [x.name, q2]) + [q3 - q1] + row_xtile)
        last = x.rank
    table = texttable.Texttable(200)
315 table.set_cols_dtype(['t', 't', 't', 't', 't', 't', 't', 't', 't', 't'])
    table.set_cols_align(['r', 'l', 'r', 'r', 'c', 'r', 'r', 'r', 'r', 'r'])
    table.set_deco(texttable.Texttable.HEADER)
    table.add_rows(rows)
    return table.draw()

```