Documentation

This version of the document is dated 2020-08-17.

```
Help on module randomgen:

NAME
    randomgen

DESCRIPTION
    Sample code for the article "Randomization and Sampling Methods"
    [https://www.codeproject.com/Articles/1190459/Random-Number-Generation-Methods]
(https://www.codeproject.com/Articles/1190459/Random-Number-Generation-Methods)

    Written by Peter O.
    Any copyright is released to the Public Domain.
    [https://creativecommons.org/publicdomain/zero/1.0/]
(https://creativecommons.org/publicdomain/zero/1.0/)

CLASSES
    builtins.object
        AlmostRandom
        BinaryExpansion
        BringmannLarsen
        ConvexPolygonSampler
        DensityInversionSampler
        DensityTiling
        FastLoadedDiceRoller
        KVectorSampler
        OptimalSampler
        PascalTriangle
        RandomGen
        RatioOfUniformsTiling
        SortedAliasMethod
        VoseAlias

    class AlmostRandom(builtins.object)
     |  Methods defined here:
     |
     |  __init__(self, randgen, list)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  choose(self)
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors defined here:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class BinaryExpansion(builtins.object)
     |  Methods defined here:
     |
     |  __init__(self, arr, zerosAtEnd=False)
     |      Binary expansion of a real number in [0, 1], initialized
     |      from an array of zeros and ones expressing the binary
     |      expansion.
     |      The first binary digit is the half digit, the second
     |      is the quarter digit, the third is the one-eighth digit,
```

```
 |          and so on.  Note that the number 1 can be
 |          expressed by passing an empty array and specifying
 |          zerosAtEnd = False, and the number 0 can be
 |          expressed by passing an empty array and specifying
 |          zerosAtEnd = True.
 |          arr - Array indicating the initial digits of the binary
 |          expansion.
 |          zerosAtEnd - Indicates whether the binary expansion
 |          is expressed as 0.xxx0000... or 0.yyy1111... (e.g., 0.1010000...
 |          vs. 0.1001111....  Default is the latter case (False).
 |
 |  entropy(self)
 |
 |  eof(self)
 |          Returns True if the end of the binary expansion was reached; False otherwise.
 |
 |  fromFloat(f)
 |          Creates a binary expansion object from a 64-bit floating-point number in the
 |          interval [0, 1].
 |
 |  fromFraction(f)
 |          Creates a binary expansion object from a fraction in the
 |          interval [0, 1].
 |
 |  get(f)
 |          Creates a binary expansion object from a fraction, 'int', or
 |          'float' in the interval [0, 1]; returns 'f' unchanged, otherwise.
 |
 |  getOrReset(f)
 |          Creates a binary expansion object from a fraction, 'int', or
 |          'float' in the interval [0, 1]; resets 'f' (calls its reset method) otherwise.
 |
 |  nextbit(self)
 |          Reads the next bit in the binary expansion.
 |
 |  reset(self)
 |          Resets this object to the first bit in the binary expansion.
 |
 |  value(self)
 |
 |  ----------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |          dictionary for instance variables (if defined)
 |
 |  __weakref__
 |          list of weak references to the object (if defined)

class BringmannLarsen(builtins.object)
 |  Implements Bringmann and Larsen's sampler, which chooses a random number in [0, n)
 |  where the probability that each number is chosen is weighted.  The 'weights' is the
 |  list of weights each 0 or greater; the higher the weight, the greater
 |  the probability.  This sampler supports only integer weights.
 |  This is a succinct (space-saving) data structure for this purpose.
 |
 |  Reference:
 |  K. Bringmann and K. G. Larsen, "Succinct Sampling from Discrete
 |  Distributions", In: Proc. 45th Annual ACM Symposium on Theory
 |  of Computing (STOC'13), 2013.
 |
 |  Methods defined here:
```

```
 |
 |  __init__(self, weights)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  next(self, randgen)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class ConvexPolygonSampler(builtins.object)
 |  A class for uniform random sampling of
 |  points from a convex polygon.  This
 |  class only supports convex polygons because
 |  the random sampling process involves
 |  triangulating a polygon, which is trivial
 |  for convex polygons only. "randgen" is a RandomGen
 |  object, and "points" is a list of points
 |  (two-item lists) that make up the polygon.
 |
 |  Methods defined here:
 |
 |  __init__(self, randgen, points)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  sample(self)
 |      Choose a random point in the convex polygon
 |      uniformly at random.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class DensityInversionSampler(builtins.object)
 |  A sampler that generates random samples from
 |    a continuous distribution for which
 |    only the probability density function (PDF) is known,
 |    using the inversion method.  This sampler
 |    allows quantiles for the distribution to be calculated
 |    from pregenerated uniform random numbers in [0, 1].
 |
 |  - pdf: A function that specifies the PDF. It takes a single
 |    number and outputs a single number. The area under
 |    the PDF need not equal 1 (this sampler works even if the
 |    PDF is only known up to a normalizing constant).
 |  - bl, br - Specifies the sampling domain of the PDF.  Both
 |     bl and br are numbers giving the domain,
 |     which in this case is [bl, br].  For best results, the
 |     probabilities outside the sampling domain should be
 |     negligible (the reference cited below uses cutoff points
 |     such that the probabilities for each tail integrate to
 |     about ures*0.05 or less).
```

```
 |   - ures - Maximum approximation error tolerable, or
 |     "u-resolution".  Default is 10^-8.  This error tolerance
 |     "does not work for continuous distributions [whose PDFs
 |     have] high and narrow peaks or poles".
 |
 |     Reference:
 |     Gerhard Derflinger, Wolfgang Hörmann, and Josef Leydold,
 |     "Random variate generation by numerical inversion when
 |     only the density is known", ACM Transactions on Modeling
 |     and Computer Simulation 20(4) article 18, October 2010.
 |
 |   Methods defined here:
 |
 |   __init__(self, pdf, bl, br, ures=1e-08)
 |       Initialize self.  See help(type(self)) for accurate signature.
 |
 |   codegen(self, name='dist')
 |       Generates standalone Python code that samples
 |               (approximately) from the distribution estimated
 |               in this class.  Idea from Leydold, et al.,
 |               "An Automatic Code Generator for
 |               Nonuniform Random Variate Generation", 2001.
 |       - name: Distribution name.  Generates Python methods called
 |           sample_X (samples one random number), and quantile_X
 |           (finds the quantile
 |           for a uniform random number in [0, 1]),
 |           where X is the name given here.
 |
 |   quantile(self, v)
 |       Calculates quantiles from uniform random numbers
 |           in the interval [0, 1].
 |       - v: A list of uniform random numbers.
 |       Returns a list of the quantiles corresponding to the
 |       uniform random numbers.  The returned list will have
 |       the same number of entries as 'v'.
 |
 |   sample(self, rg, n=1)
 |       Generates random numbers that (approximately) follow the
 |           distribution modeled by this class.
 |       - n: The number of random numbers to generate.
 |       Returns a list of 'n' random numbers.
 |
 |   ----------------------------------------------------------------------
 |   Data descriptors defined here:
 |
 |   __dict__
 |       dictionary for instance variables (if defined)
 |
 |   __weakref__
 |       list of weak references to the object (if defined)

class DensityTiling(builtins.object)
 |   Produces a tiling of a probability density function (PDF)
 |       for the purposes of random number generation.  The PDF is
 |       decomposed into tiles; these tiles will either cross the PDF
 |       or go below the PDF.  In each recursion cycle, each tile is
 |       split into four tiles, and tiles that end up above the PDF are
 |       discarded.
 |
 |   - pdf: A function that specifies the PDF. It takes a single
 |     number and outputs a single number. The area under
 |     the PDF need not equal 1 (this class tolerates the PDF even if
```

```
          |      it is only known up to a normalizing constant).  For best results,
          |      the PDF should be bounded from above (that is, it should be free of _poles_, or
points
          |      that approach infinity).  If the PDF does contain a pole, this class
          |      may accommodate the pole by sampling from a modified version of the PDF,
          |      so that points extremely close to the pole may be sampled
          |      at a higher or lower probability than otherwise (but not in a way
          |      that significantly affects the chance of sampling points
          |      outside the pole region).
          | - bl, br - Specifies the sampling domain of the PDF.  Both
          |      bl and br are numbers giving the domain,
          |      which in this case is [bl, br].
          | - cycles - Number of recursion cycles in which to split tiles
          |      that follow the PDF.  Default is 8.
          |
          |  Additional improvements not yet implemented: Hörmann et al.,
          |  "Inverse Transformed Density Rejection for Unbounded Monotone Densities", 2007.
          |
          |  Reference:
          |  Fulger, Daniel and Guido Germano. "Automatic generation of
          |  non-uniform random variates for arbitrary pointwise computable
          |  probability densities by tiling",
          |  arXiv:0902.3088v1  [cs.MS], 2009.
          |
          |  Methods defined here:
          |
          |  __init__(self, pdf, bl, br, cycles=8)
          |      Initialize self.  See help(type(self)) for accurate signature.
          |
          |  codegen(self, name, pdfcall=None)
          |      Generates Python code that samples
          |              (approximately) from the distribution estimated
          |              in this class.  Idea from Leydold, et al.,
          |              "An Automatic Code Generator for
          |              Nonuniform Random Variate Generation", 2001.
          |      - name: Distribution name.  Generates a Python method called
          |        sample_X where X is the name given here (samples one
          |        random number).
          |      - pdfcall: Name of the method representing pdf (for more information,
          |        see the __init__ method of this class).  Optional; if not given
          |        the name is pdf_X where X is the name given in the name parameter.
          |
          |  maybeAppend(self, pdfevals, newtiles, xmn, xmx, ymn, ymx)
          |
          |  sample(self, rg, n=1)
          |      Generates random numbers that (approximately) follow the
          |           distribution modeled by this class.
          |      - n: The number of random numbers to generate.
          |      Returns a list of 'n' random numbers.
          |
          |  ----------------------------------------------------------------------
          |  Data descriptors defined here:
          |
          |  __dict__
          |      dictionary for instance variables (if defined)
          |
          |  __weakref__
          |      list of weak references to the object (if defined)

    class FastLoadedDiceRoller(builtins.object)
       |  Implements the Fast Loaded Dice Roller, which chooses a random number in [0, n)
       |  where the probability that each number is chosen is weighted.  The 'weights' is the
```

```
|      list of weights each 0 or greater; the higher the weight, the greater
|      the probability.  This sampler supports only integer weights.
|
|   Reference: Saad, F.A., Freer C.E., et al. "The Fast Loaded Dice Roller: A
|   Near-Optimal Exact Sampler for Discrete Probability Distributions", in
|   _AISTATS 2020: Proceedings of the 23rd International Conference on Artificial
|   Intelligence and Statistics, Proceedings of Machine Learning Research_ 108,
|   Palermo, Sicily, Italy, 2020.
|
|   Methods defined here:
|
|   __init__(self, weights)
|        Initialize self.  See help(type(self)) for accurate signature.
|
|   codegen(self, name='sample_discrete')
|        Generates standalone Python code that samples
|               from the distribution modeled by this class.
|               Idea from Leydold, et al.,
|               "An Automatic Code Generator for
|               Nonuniform Random Variate Generation", 2001.
|        - name: Method name. Default: 'sample_discrete'.
|
|   next(self, randgen)
|
|   ----------------------------------------------------------------------
|   Data descriptors defined here:
|
|   __dict__
|        dictionary for instance variables (if defined)
|
|   __weakref__
|        list of weak references to the object (if defined)

class KVectorSampler(builtins.object)
|   A K-Vector-like sampler of a continuous distribution
|   with a known cumulative distribution function (CDF).
|   Uses algorithms
|   described in Arnas, D., Leake, C., Mortari, D., "Random
|   Sampling using k-vector", Computing in Science &
|   Engineering 21(1) pp. 94-107, 2019, and Mortari, D.,
|   Neta, B., "k-Vector Range Searching Techniques".
|
|   Methods defined here:
|
|   __init__(self, cdf, xmin, xmax, pdf=None, nd=200)
|        Initializes the K-Vector-like sampler.
|        Parameters:
|        - cdf: Cumulative distribution function (CDF) of the
|          distribution.  The CDF must be
|          monotonically nondecreasing everywhere in the
|          interval [xmin, xmax] and must output values in [0, 1];
|          for best results, the CDF should
|          be increasing everywhere in [xmin, xmax].
|        - xmin: Maximum x-value to generate.
|        - xmax: Maximum x-value to generate.  For best results,
|          the range given by xmin and xmax should cover all or
|          almost all of the distribution.
|        - pdf: Optional. Distribution's probability density
|          function (PDF), to improve accuracy in the root-finding
|          process.
|        - nd: Optional. Size of tables used in the sampler.
|          Default is 200.
```

```
 |
 |  quantile(self, uniforms)
 |      Returns a list of 'n' numbers that correspond
 |      to the given uniform random numbers and follow
 |      the distribution represented by this sampler.  'uniforms'
 |      is a list of uniform random values in the interval
 |      [0, 1].  For best results, this sampler's range
 |      (xmin and xmax in the constructor)
 |      should cover all or almost all of the desired distribution and
 |      the distribution's CDF should be monotonically
 |      increasing everywhere (every number in the distribution's
 |      range has nonzero probability of occurring), since
 |      among other things,
 |      this method maps each uniform value to the
 |      range of CDFs covered by this distribution (that is,
 |      [0, 1] is mapped to [minCDF, maxCDF]), and
 |      uniform values in "empty" regions (regions with
 |      constant CDF) are handled by replacing those
 |      values with the minimum CDF value covered.
 |
 |  sample(self, rg, n)
 |      Returns a list of 'n' random numbers of
 |      the distribution represented by this sampler.
 |      - rg: A random generator (RandGen) object.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class OptimalSampler(builtins.object)
 |  Implements a sampler which chooses a random number in [0, n)
 |  where the probability that each number is chosen is weighted.  The 'weights' is the
 |  list of weights each 0 or greater; the higher the weight, the greater
 |  the probability.  This sampler supports only integer weights, but the sampler is
 |  entropy-optimal as long as the sum of those weights is of the form 2^k or 2^k-2^m.
 |
 |  Reference: Feras A. Saad, Cameron E. Freer, Martin C. Rinard, and Vikash K.
Mansinghka.
 |  Optimal Approximate Sampling From Discrete Probability Distributions. Proc.
 |  ACM Program. Lang. 4, POPL, Article 36 (January 2020), 33 pages.
 |
 |  Methods defined here:
 |
 |  __init__(self, m)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  codegen(self, name='sample_discrete')
 |      Generates standalone Python code that samples
 |              from the distribution modeled by this class.
 |              Idea from Leydold, et al.,
 |              "An Automatic Code Generator for
 |              Nonuniform Random Variate Generation", 2001.
 |      - name: Method name. Default: 'sample_discrete'.
 |
 |  next(self, rg)
 |
 |  nextFromMatrix(self, pm, rg)
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class PascalTriangle(builtins.object)
 |  Generates the rows of Pascal's triangle, or the
 |  weight table for a binomial(n,1/2) distribution.
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  aliasinfo(self, desiredRow)
 |
 |  getrow(self, desiredRow)
 |      Calculates an arbitrary row of Pascal's triangle.
 |
 |  next(self)
 |      Generates the next row of Pascal's triangle, starting with
 |      row 0. The return value is a list of row-number-choose-k
 |      values.
 |
 |  nextto(self, desiredRow)
 |      Generates the row of Pascal's triangle with the given row number,
 |      skipping all rows in between.  The return value is a list of
 |      row-number-choose-k values.
 |
 |  row(self)
 |      Gets the row number of the row that will be generated
 |      the next time _next_ is called.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class RandomGen(builtins.object)
 |  A class that implements many methods for
 |  random number generation and sampling.  It takes
 |  an underlying RNG as specified in the constructor.
 |
 |  Methods defined here:
 |
 |  __init__(self, rng=None)
 |      Initializes a new RandomGen instance.
 |      NOTES:
 |
 |      1. Assumes that 'rng' implements
 |      a 'randint(a, b)' method that returns a random
 |      integer in the interval [a, b].  Currently, this
 |      class assumes 'a' is always 0.
```

```
 |      2. 'rndint' (and functions that ultimately call it) may be
 |      slower than desirable if many random numbers are
 |      needed at once.  Ways to improve the performance
 |      of generating many random numbers at once include
 |      vectorization (which is often PRNG specific) and multithreading
 |      (which is too complicated to show here).
 |
 |  ball_point(self, dims, radius=1)
 |      Generates an independent and uniform random point inside a 'dims'-dimensional
 |      ball (disc, solid sphere, etc.) centered at the origin.
 |
 |  bernoulli(self, p)
 |      Returns 1 at probability p, 0 otherwise.
 |
 |  beta(self, a, b, nc=0)
 |      Generates a beta-distributed random number.
 |      `a` and `b` are the two parameters of the beta distribution,
 |      and `nc` is a parameter such that `nc` other than 0
 |      indicates a _noncentral_ distribution.
 |
 |  binomial(self, trials, p, n=None)
 |
 |  binomial_int(self, trials, px, py)
 |
 |  cauchy(self)
 |
 |  choice(self, list)
 |
 |  derangement(self, list)
 |      Returns a copy of list with each of its elements
 |      moved to a different position.
 |
 |  derangement_algorithm_s(self, list)
 |      Returns a copy of 'list' with each of its elements
 |      moved to a different position (a derangement),
 |      but with the expected number of cycle lengths
 |      in probability, even though the list
 |      need not be a uniformly randomly
 |      chosen derangement.  Uses importance sampling.
 |      Reference:
 |      J.R.G. Mendonça, "Efficient generation of
 |      random derangements with the expected
 |      distribution of cycle lengths", arXiv:1809.04571v4
 |      [stat.CO], 2020.
 |
 |  derangement_algorithm_t(self, list)
 |      Returns a copy of 'list' with each of its elements
 |      moved to a different position (a derangement),
 |      but with the expected number of cycle lengths
 |      in probability, even though the list
 |      need not be a uniformly randomly
 |      chosen derangement.  Reference:
 |      J.R.G. Mendonça, "Efficient generation of
 |      random derangements with the expected
 |      distribution of cycle lengths", arXiv:1809.04571v4
 |      [stat.CO], 2020.
 |
 |  diceRoll(self, dice, sides=6, bonus=0)
 |
 |  dirichlet(alphas)
 |
 |  discretegen(self, probs)
```

| Generates a random integer in [0, n), where the probability
| of drawing each integer is specified as a list
| of probabilities that sum to 1, where n is the
| number of probabilities. This method is optimal,
| or at least nearly so, in terms of the number of random
| bits required to generate the number
| on average. This method implements
| a solution to exercise 3.4.2 of chapter 15 of Luc Devroye's
| _Non-Uniform Random Variate Generation_, 1986.
|
| - probs. List of probability objects, where for each item
| in the probability list, the integer 'i' is chosen
| with probability 'probs[i]'.
| Each probability object provides access to a binary
| expansion of the probability, which must be a real number in
| the interval [0, 1]. The binary expansion is a sequence of zeros and ones
| expressed as follows: The first binary digit is the half digit, the second
| is the quarter digit, the third is the one-eighth digit,
| and so on. Note that any probability with a terminating binary
| expansion (except 0) can be implemented by "subtracting" 1
| from the expansion and then appending an infinite sequence
| of ones at the end. The probability object must implement the following
| three methods:
| - reset(): Resets the probability object to the first digit in
| the binary expansion.
| - nextbit(): Gets the next digit in the binary expansion.
| - eof(): Gets whether the end of the binary expansion was reached
| (True or False), meaning the rest of the digits in the expansion are
| all zeros.
| The probability object will have to be mutable for this method
| to work.
| The BinaryExpansion class is a convenient way to express numbers
| as probability objects that meet these criteria. Each probability object
| can also be a float, int, or Fraction in the interval [0, 1].
|
| expoNumerator(self, denom)
| Generates the numerator of an exponential random
| number with a given denominator,
| using von Neumann's
| algorithm ("Various techniques used in connection with
| random digits", 1951).
|
| expoRatio(self, base, rx=1, ry=1)
| Generates an exponential random number
| (in the form of a ratio, or two-element list) given
| the rate `rx`/`ry` and the base `base`.
| The number will have the denominator `base*rx`.
|
| exponential(self, lamda=1.0)
|
| exprandfill(self, a, bits)
| Fills the unsampled bits of the given exponential random number
| 'a' as necessary to make a number whose fractional part
| has 'bits' many bits. If the number's fractional part already has
| that many bits or more, the number is rounded using the round-to-nearest,
| ties to even rounding rule. Returns the resulting number as a
| multiple of 2^'bits'.
|
| exprandless(self, a, b)
| Determines whether one partially-sampled exponential number
| is less than another; returns
| True if so and False otherwise. During

```
     |      the comparison, additional bits will be sampled in both numbers
     |      if necessary for the comparison.
     |
     |  exprandnew(self, lamdanum=1, lamdaden=1)
     |      Returns an object to serve as a partially-sampled
     |      exponential random number with the given
     |      rate 'lamdanum'/'lamdaden'.  The object is a list of five numbers:
     |      the first is a multiple of 1/(2^X), the second is X, the third is the integer
     |      part (initially -1 to indicate the integer part wasn't sampled yet),
     |      and the fourth and fifth are the lamda parameter's
     |      numerator and denominator, respectively.  Default for 'lamdanum'
     |      and 'lamdaden' is 1.
     |      The number created by this method will be "empty"
     |      (no bits sampled yet).
     |
     |  frechet(self, a, b, mu=0)
     |
     |  gamma(self, mean, b=1.0, c=1.0, d=0.0)
     |      Generates a random number following a gamma distribution.
     |
     |  gaussian_copula(self, cov)
     |
     |  geoellipsoid_point(self, a=6378.137, invf=298.2572236)
     |      Generates an independent and uniform random
     |      point on the surface of a geoellipsoid.  The
     |      geoellipsoid uses the following parameters:
     |      a - semimajor axis (distance from the center of
     |         the geoellipsoid to the equator).  The default
     |         is the WGS 84 ellipsoid's semimajor axis
     |         in kilometers.
     |      invf - inverse flattening.  The default is the
     |         WGS 84 ellipsoid's inverse flattening.
     |
     |  geometric(self, p)
     |
     |  gumbel(self, a, b)
     |
     |  hypercube_point(self, dims, sizeFromCenter=1)
     |      Generates an independent and uniform random point on the surface of a 'dims'-
dimensional
     |      hypercube (square, cube, etc.)
     |      centered at the origin.
     |
     |  hypergeometric(self, trials, ones, count)
     |
     |  hypersphere_point(self, dims, radius=1)
     |      Generates an independent and uniform random point on the surface of a 'dims'-
dimensional
     |      hypersphere (circle, sphere, etc.)
     |      centered at the origin.
     |
     |  integersWithSum(self, n, total)
     |      Returns a list of 'n' integers 0 or greater that sum to 'total'.
     |      The combination is chosen uniformly at random among all
     |      possible combinations.
     |
     |  integers_from_pdf(self, pdf, mn, mx, n=1)
     |      Generates one or more random integers from a discrete probability
     |      distribution expressed as a probability density
     |      function (PDF), which is also called the probability mass
     |      function for discrete distributions.  The random integers
     |      will be in the interval [mn, mx].  `n` random integers will be
```

```
 |      generated. `pdf` is the PDF; it takes one parameter and returns,
 |      for that parameter, a weight indicating the relative likelihood
 |      that a random integer will equal that parameter.
 |      The area under the "curve" of the PDF need not be 1.
 |      By default, `n` is 1.
 |
 |  integers_from_u01(self, u01, pmf)
 |      Transforms one or more random numbers into numbers
 |      (called quantiles) that
 |      follow a discrete distribution, assuming the distribution
 |          produces only integers 0 or greater.
 |          - `u01` is a list of uniform random numbers, in [0, 1].
 |          - `pmf` is the probability mass function (PMF)
 |          of the discrete distribution; it takes one parameter and returns,
 |          for that parameter, the probability that a random number is
 |          equal to that parameter (each probability is in the interval [0, 1]).
 |          The area under the PMF must be 1; it
 |          is not enough for the PMF to be correct up to a constant.
 |
 |  intsInRangeSortedWithSum(self, numSamples, numPerSample, mn, mx, sum)
 |      Generates one or more combinations of
 |       'numPerSample' numbers each, where each
 |       combination's numbers sum to 'sum' and are listed
 |       in sorted order, and each
 |       number is in the interval '[mn, mx]'.
 |        The combinations are chosen uniformly at random.
 |          'mn', 'mx', and
 |       'sum' may not be negative.  Returns an empty
 |       list if 'numSamples' is zero.
 |        The algorithm is thanks to a _Stack Overflow_
 |      answer (`questions/61393463`) by John McClane.
 |      Raises an error if there is no solution for the given
 |      parameters.
 |
 |  intsInRangeWithSum(self, numSamples, numPerSample, mn, mx, sum)
 |      Generates one or more combinations of
 |       'numPerSample' numbers each, where each
 |       combination's numbers sum to 'sum' and are listed
 |       in any order, and each
 |       number is in the interval '[mn, mx]'.
 |        The combinations are chosen uniformly at random.
 |          'mn', 'mx', and
 |       'sum' may not be negative.  Returns an empty
 |       list if 'numSamples' is zero.
 |        The algorithm is thanks to a _Stack Overflow_
 |      answer (`questions/61393463`) by John McClane.
 |      Raises an error if there is no solution for the given
 |      parameters.
 |
 |  intsInRangesWithSum(self, numSamples, ranges, total)
 |      Generates one or more combinations of
 |       'len(ranges)' numbers each, where each
 |       combination's numbers sum to 'total', and each number
 |       has its own valid range.  'ranges' is a list of valid ranges
 |       for each number; the first item in each range is the minimum
 |       value and the second is the maximum value.  For example,
 |       'ranges' can be [[1,4],[3,5],[2,6]], which says that the first
 |       number must be in the interval [1, 4], the second in [3, 5],
 |       and the third in [2, 6].
 |        The combinations are chosen uniformly at random.
 |          Neither the integers in the 'ranges' list nor
 |       'total' may be negative.  Returns an empty
```

```
|          list if 'numSamples' is zero.
|           This is a modification I made to an algorithm that
|             was contributed in a _Stack Overflow_
|        answer (`questions/61393463`) by John McClane.
|        Raises an error if there is no solution for the given
|        parameters.
|
| kth_smallest_of_n_u01(self, k, n)
|        Generates the kth smallest number among n random numbers
|        in the interval [0, 1].
|
| kthsmallest(self, n, k, b)
|        Generates the 'k'th smallest 'b'-bit uniform random
|        number out of 'n' of them.
|
| kthsmallest_urand(self, n, k)
|        Generates the 'k'th smallest 'b'-bit uniform random
|        number out of 'n' of them; returns the result in
|        the form of a "u-rand", or a partially sampled uniform
|        random number (Karney, "Sampling exactly from the normal distribution").
|
| latlon(self)
|        Generates an independent and uniform random latitude and
|        longitude, in radians.  West and south coordinates
|        are negative.
|
| lognormal(self, mu=0.0, sigma=0.0)
|
| lower_bound_copula(self)
|
| mcmc(self, pdf, n)
|        Generates 'n' random numbers that follow
|        the probability density given in 'pdf' using
|        a Markov-chain Monte Carlo algorithm, currently
|        Metropolis--Hastings.  The resulting random numbers
|        are not independent, but are often close to
|        being independent.  'pdf' takes one number as
|        a parameter and returns a number 0 or greater.
|        The area under the curve (integral) of 'pdf'
|        need not be equal to 1.
|
| mcmc2(self, pdf, n)
|        Generates 'n' pairs of random numbers that follow
|        the probability density given in 'pdf' using
|        a Markov-chain Monte Carlo algorithm, currently
|        Metropolis--Hastings.  The resulting random pairs
|        are not independent, but are often close to
|        being independent.  'pdf' takes one parameter,
|        namely, a list of two numbers giving a sampled
|        point and returns a number 0 or greater.
|        The volume under the surface (integral) of 'pdf'
|        need not be equal to 1.
|
| monte_carlo_integrate(self, func, bounds, samples=1000)
|        Estimates the integral (volume) of a function within the
|        given bounds using Monte Carlo integration, which generates
|        an estimate using the help of randomization.
|        func - Function to integrate.  Takes the same number
|           of parameters as the length of bounds.
|        bounds - Bounds of integration at each dimension.
|           An N-length array of arrays.  Each array in turn
|           contains two items: the lower bound and upper bound
```

```
|           for that dimension.
|       samples - Number of times to sample the bounds of
|           integration randomly.  The default is 1000 samples.
|       Returns an array containing two items: the estimated
|       integral and the standard error.
|
|  moyal(self, mu=0, sigma=1)
|       Sample from a Moyal distribution, using the
|       method given in C. Walck, "Handbook on
|       Statistical Distributions for Experimentalists",
|       pp. 93-94.
|
|  multinomial(self, trials, weights)
|
|  multinormal(self, mu, cov)
|
|  multinormal_n(self, mu, cov, n=1)
|
|  multipoisson(self, firstmean, othermeans)
|       Multivariate Poisson distribution (as found in Mathematica).
|
|  multivariate_t(self, mu, cov, df)
|       Multivariate t-distribution, mu is the mean (can be None),
|       cov is the covariance matrix, and df is the degrees of freedom.
|
|  negativeMultinomial(self, succ, failures)
|       Negative multinomial distribution.
|
|       Models the number of failures of one or more
|       kinds before a given number of successes happens.
|       succ: Number of successes.
|       failures: Contains probabilities for each kind of failure.
|       The sum of probabilities must be less than 1.
|       Returns: A list containing a random number
|       of failures of each kind of failure.
|
|  negativebinomial(self, successes, p)
|
|  negativebinomialint(self, successes, px, py)
|       Generates a negative binomial random number, defined
|       here as the number of failures before 'successes' many
|       successful trials, where the probability of success in
|       each trial is px/py.
|
|  nonzeroIntegersWithSum(self, n, total)
|       Returns a list of 'n' integers greater than 0 that sum to 'total'.
|       The combination is chosen uniformly at random among all
|       possible combinations.
|
|  normal(self, mu=0.0, sigma=1.0)
|       Generates a normally-distributed random number.
|
|  numbersWithSum(self, count, sum=1.0)
|
|  numbers_from_cdf(self, cdf, mn, mx, n=1)
|       Generates one or more random numbers from a continuous probability
|       distribution by numerically inverting its cumulative
|       distribution function (CDF).
|
|       - cdf: The CDF; it takes one parameter and returns,
|       for that parameter, the probability that a random number will
|       be less than or equal to that parameter.
```

```
       |       - mn, mx: Sampling domain.  The random number
       |         will be in the interval [mn, mx].
       |       - n: How many random numbers to generate. Default is 1.
       |
       |  numbers_from_dist(self, pdf, mn=0, mx=1, n=1, bitplaces=53)
       |       Generates 'n' random numbers that follow a continuous
       |       distribution in an interval [mn, mx].  The distribution's
       |       PDF (probability density function) must be bounded from above
       |       (have a finite value) and be continuous almost everywhere
       |       in the interval.  Implements section 4 of Devroye and Gravel,
       |       "The expected bit complexity of the von Neumann rejection
       |       algorithm", arXiv:1511.02273v2  [cs.IT], 2016.
       |       - 'n' is the number of random numbers to generate.  Default is 1.
       |       - 'pdf' is a procedure that takes three arguments: xmin, xmax, bitplaces,
       |          and returns an array of two items: the greatest lower bound of f(x) anywhere
       |          in the interval [xmin, xmax] (where f(x) is the PDF), and the least upper
       |          bound of f(x) anywhere there.  Both bounds are multiples of 2^-bitplaces.
       |       - 'bitplaces' is an accuracy expressed as a number of bits after the
       |          binary point. The random number will be a multiple of 2^-bitplaces,
       |          or have a smaller granularity. Default is 53.
       |       - 'mn' and 'mx' express the interval.  Both are optional and
       |          are set to 0 and 1, respectively, by default.
       |
       |  numbers_from_dist_inversion(self, icdf, n=1, digitplaces=53, base=2)
       |       Generates 'n' random numbers that follow a continuous
       |       or discrete probability distribution, using the inversion method.
       |       Implements section 5 of Devroye and Gravel,
       |       "Sampling with arbitrary precision", arXiv:1502.02539v5 [cs.IT], 2015.
       |       - 'n' is the number of random numbers to generate.  Default is 1.
       |       - 'icdf' is a procedure that takes three arguments: u, ubits, digitplaces,
       |          and returns a number within base^-digitplaces of the True inverse
       |          CDF (inverse cumulative distribution function, or quantile function)
       |          of u/base^ubits, and is monotonic for a given value of `digitplaces`.
       |       - 'digitplaces' is an accuracy expressed as a number of digits after the
       |          point. Each random number will be a multiple of base^-digitplaces,
       |          or have a smaller granularity. Default is 53.
       |       - base is the digit base in which the accuracy is expressed. Default is 2
       |          (binary). (Note that 10 means decimal.)
       |
       |  numbers_from_pdf(self, pdf, mn, mx, n=1, steps=100)
       |       Generates one or more random numbers from a continuous probability
       |       distribution expressed as a probability density
       |       function (PDF).  The random number
       |       will be in the interval [mn, mx].  `n` random numbers will be
       |       generated. `pdf` is the PDF; it takes one parameter and returns,
       |       for that parameter, a weight indicating the relative likelihood
       |        that a random number will be close to that parameter. `steps`
       |       is the number of subintervals between sample points of the PDF.
       |       The area under the curve of the PDF need not be 1.
       |       By default, `n` is 1 and `steps` is 100.
       |
       |  numbers_from_u01(self, u01, pdf, cdf, mn, mx, ures=None)
       |       Transforms one or more random numbers into numbers
       |       (called quantiles) that follow a continuous probability distribution, based on
its PDF
       |       (probability density function) and/or its CDF (cumulative distribution
       |       function).
       |
       |       - u01: List of uniform random numbers in [0, 1] that will be
       |       transformed into numbers that follow the distribution.
       |       - pdf: The PDF; it takes one parameter and returns,
       |       for that parameter, the relative probability that a
```

| random number close to that number is chosen. The area under
| the PDF need not be 1 (this method works even if the PDF
| is only known up to a normalizing constant). Optional if a CDF is given.
| - cdf: The CDF; it takes one parameter and returns,
| for that parameter, the probability that a random number will
| be less than or equal to that parameter. Optional if a PDF is given.
| For best results, the CDF should be
| monotonically nondecreasing everywhere in the
| interval [xmin, xmax] and must output values in [0, 1];
| for best results, the CDF should
| be increasing everywhere in [xmin, xmax].
| - mn, mx: Sampling domain. The random number
| will be in the interval [mn, mx]. For best results,
| the range given by mn and mx should cover all or
| almost all of the distribution.
| - ures - Maximum approximation error tolerable, or
| "u-resolution". Default is 10^-8. Currently used only if a
| PDF is given.
|
| pareto(self, minimum, alpha)
|
| partialshuffle(self, list, k)
| Does a partial shuffle of
| a list's items (stops when 'k' items
| are shuffled); the shuffled items
| will appear at the end of the list.
| Returns 'list'.
|
| piecewise_linear(self, values, weights)
|
| piecewise_linear_n(self, values, weights, n=1)
|
| poisson(self, mean)
| Generates a random number following a Poisson distribution.
|
| poissonint(self, mx, my)
| Generates a random number following a Poisson distribution with mean mx/my.
|
| polya_int(self, sx, sy, px, py)
| Generates a negative binomial (Polya) random number, defined
| here as the number of failures before 'successes' many
| successful trials (sx/sy), where the probability of success in
| each trial is px/py.
|
| powerlognormal(self, p, sigma=1.0)
| Power lognormal distribution, as described in NIST/SEMATECH
| e-Handbook of Statistical Methods, [http://www.itl.nist.gov/div898/handbook/,]
(http://www.itl.nist.gov/div898/handbook/,)
| accessed Jun. 9, 2018, sec. 1.3.6.6.14.
|
| powernormal(self, p)
| Power normal distribution, as described in NIST/SEMATECH
| e-Handbook of Statistical Methods, [http://www.itl.nist.gov/div898/handbook/,]
(http://www.itl.nist.gov/div898/handbook/,)
| accessed Jun. 9, 2018, sec. 1.3.6.6.13.
|
| product_copula(self, n=2)
|
| quantile_urands(self, icdf, urands, digitplaces=53)
| Finds the quantile of 'n' uniform random numbers expressed as "u-rands", or
partially-sampled uniform random numbers (Karney, "Sampling exactly from the normal
distribution"). Implements section 5 of Devroye and Gravel, "Sampling with arbitrary

precision", arXiv:1502.02539v5 [cs.IT], 2015.
       | - 'urands' is a list of "u-rands", or partially-sampled uniform random numbers.
Each u-rand is a list of two items, namely a multiple of 1/2^X, followed by X. For example,
the following generates a list of five empty
       | u-rands: `[[0,0] for i in range(5)]`.
       | - 'icdf' is a procedure that takes three arguments: u, ubits, digitplaces,
       | and returns a number within 2^-digitplaces of the True inverse
       | CDF (inverse cumulative distribution function, or quantile function)
       | of u/2^ubits, and is monotonic for a given value of `digitplaces`.
       | - 'digitplaces' is an accuracy expressed as a number of bits after the
       | point. Each quantile will be a multiple of 2^-digitplaces,
       | or have a smaller granularity. Default is 53.
       |
       | Example: The following example generates the maximum of 10
       | random numbers, to an accuracy of 2^53.
       |
       | ur=randgen.kthsmallest_urand(10, 10)
       | maxrand=randgen.quantile_urands(icdf, [ur], 53)[0]
       |
       | randbit(self)
       |
       | randbits(self, n)
       | Generates an n-bit random integer.
       |
       | randomwalk_posneg1(self, n)
       | Random walk of uniform positive and negative steps.
       |
       | randomwalk_u01(self, n)
       | Random walk of uniform 0-1 random numbers.
       |
       | rayleigh(self, a)
       | Generates a random number following a Rayleigh distribution.
       |
       | rndint(self, maxInclusive)
       |
       | rndint_fastdiceroller(self, maxInclusive)
       |
       | rndintexc(self, maxExclusive)
       |
       | rndintexcrange(self, minInclusive, maxExclusive)
       |
       | rndintrange(self, minInclusive, maxInclusive)
       |
       | rndrange(self, minInclusive, maxInclusive)
       |
       | rndrangemaxexc(self, minInclusive, maxExclusive)
       |
       | rndrangeminexc(self, mn, mx)
       |
       | rndrangeminmaxexc(self, mn, mx)
       |
       | rndu01(self)
       |
       | rndu01oneexc(self)
       |
       | rndu01zeroexc(self)
       |
       | rndu01zerooneexc(self)
       |
       | sample(self, list, k)
       |
       | sattolo(self, list)

```
        |       Puts the elements of 'list' in random order, choosing
        |       from among all cyclic permutations (Sattolo's algorithm).
        |       Returns 'list'.
        |
        |  shell_point(self, dims, outerRadius=1, innerRadius=0.5)
        |       Generates an independent and uniform random point inside a 'dims'-dimensional
        |       spherical shell (donut, hollow sphere, etc.)
        |       centered at the origin.
        |
        |  shuffle(self, list)
        |       Puts the elements of 'list' in random order (does an
        |       in-place shuffle).  Returns 'list'.
        |
        |  simplex_point(self, points)
        |       Generates an independent and uniform random point on the surface of an N-
dimensional
        |       simplex (line segment, triangle, tetrahedron, etc.)
        |       with the given coordinates.
        |
        |  slicesample(self, pdf, n, xstart=0.1)
        |       Slice sampling of R. M. Neal.
        |       Generates 'n' random numbers that follow
        |       the probability density given in 'pdf' using
        |       slice sampling.  The resulting random numbers
        |       are not independent, but are often close to
        |         being independent.  'pdf' takes one number as
        |         a parameter and returns a number 0 or greater.
        |         The area under the curve (integral) of 'pdf'
        |         need not be equal to 1. 'xstart' should be
        |       chosen such that `pdf(xstart)>0`.
        |
        |  spsa_minimize(self, func, guess, iterations=200, constrain=None, a=None, c=None,
acap=None)
        |       Tries to find a choice of parameters that minimizes the value
        |       of a scoring function, also called the objective function or loss
        |       function, starting from an initial guess.  This method uses an
        |       algorithm called "simultaneous perturbation
        |       stochastic approximation", which is a randomized
        |       search for the minimum value of the objective function.
        |       func - Objective function, a function that calculates a score for the
        |        given array of parameters and returns that score.  The score is a
        |        single number; the lower the score, the better.
        |        The score can be negative.  (Note that the problem of maximizing
        |        the score is the same as minimizing it except
        |        that the score's sign is reversed at the end.)
        |       guess - Initial guess for the best choice of parameters.  This is an
        |        array of parameters, each of which is a number. This array has
        |        as many items as the array passed to 'func'.
        |       iterations - Maximum number of iterations in which to run the
        |        optimization process.  Default is 200.
        |       constrain - Optional. A function that takes the given array of
        |        parameters and constrains them to fit the bounds of a valid
        |        array of parameters. This function modifies the array in place.
        |       a - Optional.  A setting used in the optimization process; greater than 0.
        |       c - Optional.  A setting used in the optimization process; greater than 0. As a
guideline,
        |         'c' is about equal to the "standard deviation of the measurement noise"
        |         for several measurements at the initial guess, and is a "small positive
        |         number" if measurements are noise-free (Spall 1998).  Default
        |         is 0.001.
        |       acap - Optional.  A setting used in the optimization process; an
        |         integer greater than 0.
```

```
 |
 |  stable(self, alpha, beta)
 |      Generates a random number following a stable distribution.
 |
 |  stable0(self, alpha, beta, mu=0, sigma=1)
 |      Generates a random number following a 'type 0' stable distribution.
 |
 |  surface_point(self, f, bounds, ngrad, gmax)
 |      Generates a uniform random point on
 |        a parametric surface, using a rejection
 |        approach developed by Williamson, J.F.,
 |        "Random selection of points distributed on
 |         curved surfaces", Physics in Medicine & Biology 32(10), 1987.
 |      - f: Takes two parameters (u and v) and returns
 |        a 3-element array expressing
 |        a 3-dimensional position at the given point.
 |      - bounds: Two 2-element arrays expressing bounds
 |        for u and v.  Of the form [[umin, umax], [vmin,
 |        vmax]].
 |      - ngrad: Takes two parameters (u and v) and returns
 |        the norm of the gradient (stretch factor)
 |        at the given point.  Can be None, in which
 |        the norm-of-gradient is calculated numerically.
 |      - gmax: Maximum norm-of-gradient
 |        for entire surface.
 |
 |  t_copula(self, cov, df)
 |      Multivariate t-copula. 'cov' is the covariance matrix
 |      and 'df' is the degrees of freedom.
 |
 |  triangular(self, startpt, midpt, endpt)
 |
 |  truncnormal(randgen, a, b)
 |      Samples from a truncated normal distribution in [a, b]; this method is
 |      designed to sample from either tail of that distribution.
 |
 |      Reference:
 |      Botev, Z. and L'Ecuyer, P., 2019. Simulation from the Tail of the
 |      Univariate and Multivariate Normal Distribution. In _Systems
 |      Modeling: Methodologies and Tools_ (pp. 115-132). Springer, Cham.
 |
 |  upper_bound_copula(self, n=2)
 |
 |  vonmises(self, mean, kappa)
 |
 |  weibull(self, a, b)
 |      Generates a Weibull-distributed random number.
 |
 |  weighted_choice(self, weights)
 |
 |  weighted_choice_n(self, weights, n=1)
 |
 |  wiener(self, st, en, step=1.0, mu=0.0, sigma=1.0)
 |      Generates random numbers following a Wiener
 |      process (Brownian motion). Each element of the return
 |      value contains a timestamp and a random number in that order.
 |
 |  zero_or_one(self, px, py)
 |      Returns 1 at probability px/py, 0 otherwise.
 |
 |  zero_or_one_exp_minus(self, x, y)
 |      Generates 1 with probability exp(-px/py); 0 otherwise.
```

```
 |      Reference:
 |          Canonne, C., Kamath, G., Steinke, T., "The Discrete Gaussian
 |          for Differential Privacy", arXiv:2004.00010v2 [cs.DS], 2020.
 |
 |  zero_or_one_power(self, px, py, n)
 |      Generates 1 with probability (px/py)^n (where n can be positive, negative, or
zero); 0 otherwise.
 |
 |  zero_or_one_power_ratio(self, px, py, nx, ny)
 |      Generates 1 with probability (px/py)^(nx/ny) (where nx/ny can be positive,
negative, or zero); 0 otherwise.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  FPPRECISION = 53
 |
 |  FPRADIX = 2
 |
 |  MINEXPONENT = -1074

class RatioOfUniformsTiling(builtins.object)
 |  Produces a tiling for the purposes
 |       of fast sampling from a probability distribution via the
 |       ratio of uniforms method.
 |
 |  - pdf: The probability density function (PDF); it takes one parameter and returns,
 |     for that parameter, the relative probability that a
 |     random number close to that number is chosen.  The area under
 |     the PDF need not be 1; this method works even if the PDF
 |     is only known up to a normalizing constant, and even if
 |     the distribution has infinitely extending tails to the left and/or right.
 |     However, for the ratio of uniforms method to work, both pdf(x) and
 |     x*x*pdf(x) must be bounded from above (thus, if the distribution has
 |     tails, they must drop off at a faster than quadratic rate).
 |  - mode: X-coordinate of the PDF's highest peak or one of them,
 |     or a location close to it.  Optional; default is 0.
 |  - y0, y1: Bounding coordinates for the ratio-of-uniforms tiling.
 |     For this class to work, y0 <= min( x*sqrt(pdf(x)) ) and
 |     y1 >= max( x*sqrt(pdf(x)) ) for all x.  Optional; the default is y0=-10, y1=10.
 |  - cycles - Number of recursion cycles in which to split tiles
 |     for the ratio-of-uniforms tiling.  Default is 8.
 |
 |   Additional improvements not yet implemented:
 |   Generalized ratio-of-uniforms in Hörmann et al., "Automatic
 |   Nonuniform Random Variate Generation", 2004.
 |
 |   References:
 |   Section IV.7 of Devroye, L., "Non-Uniform Random Variate Generation", 1986.
 |   Section 4.5 of Fulger, D., "From phenomenological modelling of anomalous
 |   diffusion through continuous-time random walks and fractional
 |   calculus to correlation analysis of complex systems", dissertation,
 |   Philipps-Universität Marburg, 2009.
```

```
 |
 |  Methods defined here:
 |
 |  __init__(self, pdf, mode=0, y0=-10, y1=10, cycles=8)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  codegen(self, name, pdfcall=None)
 |      Generates Python code that samples
 |              (approximately) from the distribution estimated
 |              in this class.  Idea from Leydold, et al.,
 |              "An Automatic Code Generator for
 |              Nonuniform Random Variate Generation", 2001.
 |      - name: Distribution name.  Generates a Python method called
 |        sample_X where X is the name given here (samples one
 |        random number).
 |      - pdfcall: Name of the method representing pdf (for more information,
 |        see the __init__ method of this class).  Optional; if not given
 |        the name is pdf_X where X is the name given in the name parameter.
 |
 |  maybeAppend(self, newtiles, xmn, xmx, ymn, ymx, depth=0)
 |
 |  sample(self, rg, n=1)
 |      Generates random numbers that (approximately) follow the
 |              distribution modeled by this class.
 |      - n: The number of random numbers to generate.
 |      Returns a list of 'n' random numbers.
 |
 |  svg(self)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class SortedAliasMethod(builtins.object)
 |  Implements a weighted sampling table
 |  where each weight must be in sorted
 |  order (ascending or descending).
 |  When many entries are in the table,
 |  the initialization is faster than with
 |  FastLoadedDiceRoller or VoseAlias.  Reference:
 |  K. Bringmann and K. Panagiotou, "Efficient Sampling
 |  Methods for Discrete Distributions." In: Proc. 39th
 |  International Colloquium on Automata, Languages,
 |  and Programming (ICALP'12), 2012.
 |  -  p: List of weights, in sorted order (ascending or
 |       descending).
 |
 |  Methods defined here:
 |
 |  __init__(self, p)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  next(self, rg)
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
```

```
       |  __dict__
       |      dictionary for instance variables (if defined)
       |
       |  __weakref__
       |      list of weak references to the object (if defined)

    class VoseAlias(builtins.object)
       |  Implements Vose's version of the alias sampler, which chooses a random number in [0,
n)
       |  where the probability that each number is chosen is weighted.  The 'weights' is the
       |  list of weights each 0 or greater; the higher the weight, the greater
       |  the probability.  This sampler supports integer or non-integer weights.
       |
       |  Reference:
       |  Vose, Michael D. "A linear algorithm for generating random numbers with a given
       |  distribution." IEEE Transactions on software engineering 17, no. 9 (1991): 972-975.
       |
       |  Methods defined here:
       |
       |  __init__(self, weights)
       |      Initialize self.  See help(type(self)) for accurate signature.
       |
       |  next(self, randgen)
       |
       |  ----------------------------------------------------------------------
       |  Data descriptors defined here:
       |
       |  __dict__
       |      dictionary for instance variables (if defined)
       |
       |  __weakref__
       |      list of weak references to the object (if defined)

FUNCTIONS
    numericalTable(func, x, y, n=100)

    urandfill(rg, a, bits=53)
        Fills the unsampled bits of the given u-rand 'a' as necessary to
            make a number with 'bits' many bits.  If the u-rand already has
            that many bits or more, the u-rand is rounded using the round-to-nearest,
            ties to even rounding rule.  Returns the resulting number as a
            multiple of 2^'bits'.  Default for 'bits' is 53.
        - rg: An object that must supply a 'randbit' method that generates an
          unbiased random bit.

    urandgreater(rg, a, b)
        Determines whether the first u-rand is greater than another u-rand; returns
            True if so and False otherwise.  During
            the comparison, additional bits will be sampled in both u-rands if necessary
            for the comparison.
        - rg: An object that must supply a 'randbit' method that generates an
          unbiased random bit.
         - a, b: The first and second u-rands.

    urandless(rg, a, b)
        Determines whether the first u-rand is less than another u-rand; returns
            True if so and False otherwise.  During
            the comparison, additional bits will be sampled in both u-rands if necessary
            for the comparison.
        - rg: An object that must supply a 'randbit' method that generates an
          unbiased random bit.
         - a, b: The first and second u-rands.
```

urandnew()
        Returns an object to serve as a partially-sampled uniform random
        number called a "u-rand" (Karney, "Sampling exactly from the normal distribution").
        A u-rand is a list of two numbers: the first is a multiple of 1/(2^X), and the second
is X.
        The urand created by this method will be "empty" (no bits sampled yet).

FILE
    /home/rooster/Documents/SharpDevelopProjects/peteroupc.github.io/randomgen.py

Help on module fixed:

NAME
    fixed

CLASSES
    builtins.object
        Fixed

    class Fixed(builtins.object)
     | Fixed-point numbers, represented using integers that store multiples
     | of 2^-BITS. They are not necessarily faster than floating-point numbers, nor
     | do they necessarily have the same precision or resolution of floating-point
     | numbers. The main benefit of fixed-point numbers is that they improve
     | determinism for applications that rely on non-integer real numbers (notably
     | simulations and machine learning applications), in the sense that the operations
     | given here deliver the same answer for the same input across computers,
     | whereas floating-point numbers have a host of problems that make repeatable
     | results difficult, including differences in their implementation, rounding
     | behavior, and order of operations, as well as nonassociativity of
     | floating-point numbers.
     |
     | The operations given here are not guaranteed to be "constant-time"
     | (non-data-dependent and branchless) for all relevant inputs.
     |
     | Any copyright to this file is released to the Public Domain. In case this is not
     | possible, this file is also licensed under Creative Commons Zero version 1.0.
     |
     | Methods defined here:
     |
     | __abs__(self)
     |
     | __add__(a, b)
     |
     | __cmp__(self, other)
     |
     | __div__(a, b)
     |
     | __eq__(self, other)
     | Return self==value.
     |
     | __float__(a)
     |
     | __floordiv__(a, b)
     |
     | __ge__(self, other)
     | Return self>=value.
     |
     | __gt__(self, other)
     | Return self>value.
     |

```
 | __init__(self, i)
 |     Initialize self.  See help(type(self)) for accurate signature.
 |
 | __int__(a)
 |
 | __le__(self, other)
 |     Return self<=value.
 |
 | __lt__(self, other)
 |     Return self<value.
 |
 | __mod__(a, b)
 |
 | __mul__(a, b)
 |
 | __ne__(self, other)
 |     Return self!=value.
 |
 | __neg__(self)
 |
 | __pos__(self)
 |
 | __rdiv__(a, b)
 |
 | __repr__(self)
 |     Return repr(self).
 |
 | __rtruediv__(a, b)
 |
 | __str__(self)
 |     Return str(self).
 |
 | __sub__(a, b)
 |
 | __truediv__(a, b)
 |
 | acos(a)
 |     Calculates an approximation of the inverse cosine of the given number.
 |
 | asin(a)
 |     Calculates an approximation of the inverse sine of the given number.
 |
 | atan2(y, x)
 |     Calculates the approximate measure, in radians, of the angle formed by the
 |     X axis and a line determined by the origin and the given coordinates of a 2D
 |     point.  This is also known as the inverse tangent.
 |
 | cos(a)
 |     Calculates the approximate cosine of the given angle; the angle is in radians.
 |     For the fraction size used by this class, this method is accurate to within
 |     1 unit in the last place of the correctly rounded result for all inputs
 |     in the range [-pi*2, pi*2].
 |     This method's accuracy decreases beyond that range.
 |
 | exp(a)
 |     Calculates an approximation of e (base of natural logarithms) raised
 |     to the power of this number.  May raise an error if this number
 |     is extremely high.
 |
 | floor(a)
 |
 | log(a)
```

```
|       Calculates an approximation of the natural logarithm of this number.
|
|  pow(a, b)
|       Calculates an approximation of this number raised to the power of another number.
|
|  round(a)
|
|  sin(a)
|       Calculates the approximate sine of the given angle; the angle is in radians.
|       For the fraction size used by this class, this method is accurate to within
|       1 unit in the last place of the correctly rounded result for all inputs
|       in the range [-pi*2, pi*2].
|       This method's accuracy decreases beyond that range.
|
|  sqrt(a)
|       Calculates an approximation of the square root of the given number.
|
|  tan(a)
|       Calculates the approximate tangent of the given angle; the angle is in radians.
|       For the fraction size used by this class, this method is accurate to within
|       2 units in the last place of the correctly rounded result for all inputs
|       in the range [-pi*2, pi*2].
|       This method's accuracy decreases beyond that range.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  v(i)
|       Converts a string, integer, Decimal, or other number type into
|       a fixed-point number.  If the parameter is a Fixed, returns itself.
|       If the given number is a non-integer, returns the closest value to
|       a Fixed after rounding using the round-to-nearest-ties-to-even
|       rounding mode.  The parameter is recommended to be a string
|       or integer, and is not recommended to be a `float`.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|       dictionary for instance variables (if defined)
|
|  __weakref__
|       list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  ArcTanBitDiff = 9
|
|  ArcTanFrac = 29
|
|  ArcTanHTable = [0, 294906490, 137123709, 67461703, 33598225, 16782680,...
|
|  ArcTanTable = [421657428, 248918914, 131521918, 66762579, 33510843, 16...
|
|  BITS = 20
|
|  ExpK = 648270061
|
|  HALF = 524288
|
|  HalfPiArcTanBits = 843314856
```

```
              |
              |  HalfPiBits = 1647099
              |
              |  HalfPiHighRes = 13049665332824301121333989301986179
              |
              |  HighResFrac = 116
              |
              |  Ln2ArcTanBits = 372130559
              |
              |  Log2Bits = 726817
              |
              |  LogMin = 157286.4
              |
              |  MASK = 1048575
              |
              |  PiAndHalfHighRes = 39148995998472903364001966790595 8538
              |
              |  PiArcTanBits = 1686629713
              |
              |  PiBits = 3294199
              |
              |  PiHighRes = 26099330665648602242667977860397 2359
              |
              |  QuarterPiArcTanBits = 421657428
              |
              |  SinCosK = 326016435
              |
              |  TwoTimesPiArcTanBits = 3373259426
              |
              |  TwoTimesPiBits = 6588397
              |
              |  TwoTimesPiHighRes = 52198661331297204485335955720794 4718
              |
              |  __hash__ = None
       FILE
            /home/rooster/Documents/SharpDevelopProjects/peteroupc.github.io/fixed.py


       Help on module bernoulli:

       NAME
            bernoulli

       CLASSES
            builtins.object
                Bernoulli
                DiceEnterprise

            class Bernoulli(builtins.object)
             |  This class contains methods that generate Bernoulli random numbers,
             |      (either 1 or heads with a given probability, or 0 or tails otherwise).
             |      This class also includes implementations of so-called "Bernoulli factories",
       algorithms
             |  that turn coins biased one way into coins biased another way.
             |  Written by Peter O.
             |
             |  References:
             |  - Flajolet, P., Pelletier, M., Soria, M., "On Buffon machines and numbers",
             |  arXiv:0906.5560v2 [math.PR], 2010.
             |  - Huber, M., "Designing perfect simulation algorithms using local correctness",
             |  arXiv:1907.06748v1 [cs.DS], 2019.
             |  - Huber, M., "Nearly optimal Bernoulli factories for linear functions",
```

```
| arXiv:1308.1562v2  [math.PR], 2014.
| - Huber, M., "Optimal linear Bernoulli factories for small mean problems",
| arXiv:1507.00843v2 [math.PR], 2016.
| - Łatuszyński, K., Kosmidis, I.,  Papaspiliopoulos, O., Roberts, G.O., "Simulating
| events of unknown probabilities via reverse time martingales", arXiv:0907.4018v2
| [stat.CO], 2009/2011.
| - Goyal, V. and Sigman, K. 2012. On simulating a class of Bernstein
| polynomials. ACM Transactions on Modeling and Computer Simulation 22(2),
| Article 12 (March 2012), 5 pages.
| - Morina, G., Łatuszyński, K., et al., "From the Bernoulli Factory to a Dice
| Enterprise via Perfect Sampling of Markov Chains",
| arXiv:1912.09229v1 [math.PR], 2019.
| - Shaddin Dughmi, Jason D. Hartline, Robert Kleinberg, and Rad Niazadeh.
| 2017. Bernoulli Factories and Black-Box Reductions in Mechanism Design.
| In _Proceedings of 49th Annual ACM SIGACT Symposium on the Theory
| of Computing_, Montreal, Canada, June 2017 (STOC'17).
| - Gonçalves, F. B., Łatuszyński, K. G., Roberts, G. O. (2017).  Exact Monte
| Carlo likelihood-based inference for jump-diffusion processes.
| - Vats, D., Gonçalves, F. B., Łatuszyński, K. G., Roberts, G. O. Efficient
| Bernoulli factory MCMC for intractable likelihoods, arXiv:2004.07471v1
| [stat.CO], 2020.
| - Mendo, Luis. "An asymptotically optimal Bernoulli factory for certain
| functions that can be expressed as power series." Stochastic Processes and their
| Applications 129, no. 11 (2019): 4366-4384.
| - Canonne, C., Kamath, G., Steinke, T., "The Discrete Gaussian
| for Differential Privacy", arXiv:2004.00010v2 [cs.DS], 2020.
| - Lee, A., Doucet, A. and Łatuszyński, K., 2014. Perfect simulation using
| atomic regeneration with application to Sequential Monte Carlo,
| arXiv:1407.5770v1  [stat.CO]
|
| Methods defined here:
|
| __init__(self)
|     Creates a new instance of the Bernoulli class.
|
| add(self, f1, f2, eps=Fraction(1, 20))
|     Addition Bernoulli factory: B(p), B(q) => B(p+q) (Dughmi et al. 2017)
|     - f1, f2: Functions that return 1 if heads and 0 if tails.
|     - eps: A Fraction in (0, 1). eps must be chosen so that p+q <= 1 - eps,
|       where p and q are the probability of heads for f1 and f2, respectively.
|
| alt_series(self, f, series)
|     Alternating-series Bernoulli factory: B(p) -> B(s[0] - s[1]*p + s[2]*p^2 - ...)
|     (Łatuszyński et al. 2011).
|     - f: Function that returns 1 if heads and 0 if tails.
|     - series: Object that generates each coefficient of the series starting with the
first.
|       Each coefficient must be less than or equal to the previous and all of them
must
|       be 1 or less.
|       Implements the following two methods: reset() resets the object to the first
|       coefficient; and next() generates the next coefficient.
|
| arctan_n_div_n(self, f)
|     Arctan div N: B(p) -> B(arctan(p)/p). Uses a uniformly-fast special case of
|     the two-coin Bernoulli factory, rather than the even-parity construction in
|     Flajolet's paper, which is not uniformly fast.
|     Reference: Flajolet et al. 2010.
|      - f: Function that returns 1 if heads and 0 if tails.
|
| bernoulli_x(self, f, x)
|     Bernoulli factory with a given probability: B(p) => B(x) (Mendo 2019).
```

```
|            Mendo calls Bernoulli factories "non-randomized" if their randomness
|            is based entirely on the underlying coin.
|       - f: Function that returns 1 if heads and 0 if tails.
|       - x: Desired probability, in [0, 1].
|
| bernstein(self, f, alpha)
|       Bernstein polynomial Bernoulli factory: B(p) => B(Bernstein(alpha))
|            (Goyal and Sigman 2012).
|       - f: Function that returns 1 if heads and 0 if tails.
|       - alpha: List of Bernstein coefficients for the Bernstein polynomial,
|         whose degree is this list's length minus 1.
|          For this to work, each coefficient must be in [0, 1].
|
| coin(self, c)
|       Convenience method to generate a function that returns
|       1 (heads) with the given probability c (which must be in [0, 1])
|       and 0 (tails) otherwise.
|
| complement(self, f)
|       Complement (NOT): B(p) => B(1-p) (Flajolet et al. 2010)
|       - f: Function that returns 1 if heads and 0 if tails.
|
| conditional(self, f1, f2, f3)
|       Conditional: B(p), B(q), B(r) => B((1-r)*q+r*p)  (Flajolet et al. 2010)
|       - f1, f2, f3: Functions that return 1 if heads and 0 if tails.
|
| cos(self, f)
|       Cosine Bernoulli factory: B(p) => B(cos(p)).  Special
|       case of Algorithm3 of reverse-time martingale paper.
|
| disjunction(self, f1, f2)
|       Disjunction (OR): B(p), B(q) => B(p+q-p*q) (Flajolet et al. 2010)
|       - f1, f2: Functions that return 1 if heads and 0 if tails.
|
| divoneplus(self, f)
|       Divided by one plus p: B(p) => B(1/(1+p)), implemented
|                as a special case of the two-coin construction.  Prefer over even-parity
|                for being uniformly fast.
|       - f: Function that returns 1 if heads and 0 if tails.
|       Note that this function is slow as the probability of heads approaches 1.
|
| eps_div(self, f, eps)
|       Bernoulli factory as follows: B(p) -> B(eps/p) (Lee et al. 2014).
|       - f: Function that returns 1 if heads and 0 if tails.
|       - eps: Fraction in (0, 1), must be chosen so that eps < p, where p is
|         the probability of heads.
|
| evenparity(self, f)
|       Even parity: B(p) => B(1/(1+p)) (Flajolet et al. 2010)
|       - f: Function that returns 1 if heads and 0 if tails.
|       Note that this function is slow as the probability of heads approaches 1.
|
| exp_minus(self, f)
|       Exp-minus Bernoulli factory: B(p) -> B(exp(-p)) (Łatuszyński et al. 2011).
|       - f: Function that returns 1 if heads and 0 if tails.
|
| exp_minus_ext(self, f, c=0)
|       Extension to the exp-minus Bernoulli factory of (Łatuszyński et al. 2011):
|       B(p) -> B(exp(-p - c))
|       To the best of my knowledge, I am not aware
|                of any article or paper that presents this particular
|                Bernoulli factory (before my articles presenting
```

```
 |           accurate beta and exponential generators).
 |       - f: Function that returns 1 if heads and 0 if tails.
 |       - c: Integer part of exp-minus.  Default is 0.
 |
 |  fill_geometric_bag(self, bag, precision=53)
 |
 |  geometric_bag(self, u)
 |      Bernoulli factory for a uniformly-distributed random number in (0, 1)
 |      (Flajolet et al. 2010).
 |      - u: List that holds the binary expansion, from left to right, of the uniformly-
 |        distributed random number.  Each element of the list is 0, 1, or None (meaning
 |        the digit is not yet known).  The list may be expanded as necessary to put
 |        a new digit in the appropriate place in the binary expansion.
 |
 |  linear(self, f, cx, cy=1, eps=Fraction(1, 20))
 |      Linear Bernoulli factory: B(p) => B((cx/cy)*p) (Huber 2016).
 |      - f: Function that returns 1 if heads and 0 if tails.
 |      - cx, cy: numerator and denominator of c; the probability of heads (p) is
 multiplied
 |        by c. c must be 0 or greater. If c > 1, c must be chosen so that c*p <= 1 -
 eps.
 |      - eps: A Fraction in (0, 1). If c > 1, eps must be chosen so that c*p <= 1 - eps.
 |
 |  linear_lowprob(self, f, cx, cy=1, m=Fraction(249, 500))
 |      Linear Bernoulli factory which is faster if the probability of heads is known
 |          to be less than half: B(p) => B((cx/cy)*p) (Huber 2016).
 |      - f: Function that returns 1 if heads and 0 if tails.
 |      - cx, cy: numerator and denominator of c; the probability of heads (p) is
 multiplied
 |        by c. c must be 0 or greater. If c > 1, c must be chosen so that c*p <= m <
 1/2.
 |      - m: A Fraction in (0, 1/2). If c > 1, m must be chosen so that c*p <= m < 1/2.
 |
 |  linear_power(self, f, cx, cy=1, i=1, eps=Fraction(1, 20))
 |      Linear-and-power Bernoulli factory: B(p) => B((p*cx/cy)^i) (Huber 2019).
 |      - f: Function that returns 1 if heads and 0 if tails.
 |      - cx, cy: numerator and denominator of c; the probability of heads (p) is
 multiplied
 |        by c. c must be 0 or greater. If c > 1, c must be chosen so that c*p <= 1 -
 eps.
 |      - i: The exponent.  Must be an integer and 0 or greater.
 |      - eps: A Fraction in (0, 1). If c > 1, eps must be chosen so that c*p <= 1 - eps.
 |
 |  logistic(self, f, cx=1, cy=1)
 |      Logistic Bernoulli factory: B(p) -> B(cx*p/(cy+cx*p)) or
 |          B(p) -> B((cx/cy)*p/(1+(cx/cy)*p)) (Morina et al. 2019)
 |      - f: Function that returns 1 if heads and 0 if tails.  Note that this function
 can
 |        be slow as the probability of heads approaches 0.
 |      - cx, cy: numerator and denominator of c; the probability of heads (p) is
 multiplied
 |        by c. c must be in (0, 1).
 |
 |  mean(self, f1, f2)
 |      Mean: B(p), B(q) => B((p+q)/2)  (Flajolet et al. 2010)
 |      - f1, f2: Functions that return 1 if heads and 0 if tails.
 |
 |  old_linear(self, f, cx, cy=1, eps=Fraction(1, 20))
 |      Linear Bernoulli factory: B(p) => B((cx/cy)*p). Older algorithm given in (Huber
 2014).
 |      - f: Function that returns 1 if heads and 0 if tails.
 |      - cx, cy: numerator and denominator of c; the probability of heads (p) is
```

```
multiplied
        |              by c. c must be 0 or greater. If c > 1, c must be chosen so that c*p < 1 - eps.
        |          - eps: A Fraction in (0, 1). If c > 1, eps must be chosen so that c*p < 1 - eps.
        |
        |  one_div_pi(self)
        |      Generates 1 with probability 1/pi.
        |      Reference: Flajolet et al. 2010.
        |
        |  power(self, f, ax, ay=1)
        |      Power Bernoulli factory: B(p) => B(p^(ax/ay)). (case of (0, 1) provided by
        |       Mendo 2019).
        |      - f: Function that returns 1 if heads and 0 if tails.
        |      - ax, ay: numerator and denominator of the desired power to raise the probability
        |       of heads to. This power must be 0 or greater.
        |
        |  powerseries(self, f)
        |      Power series Bernoulli factory: B(p) => B(1 - c(0)*(1-p) + c(1)*(1-p)^2 +
        |       c(2)*(1-p)^3 + ...), where c(i) = `c[i]/sum(c)`) (Mendo 2019).
        |      - f: Function that returns 1 if heads and 0 if tails.
        |      - c: List of coefficients in the power series, all of which must be
        |       non-negative integers.
        |
        |  probgenfunc(self, f, rng)
        |      Probability generating function Bernoulli factory: B(p) => B(E[p^x]), where x is
rng()
        |       (Dughmi et al. 2017). E[p^x] is the expected value of p^x and is also known
        |       as the probability generating function.
        |      - f: Function that returns 1 if heads and 0 if tails.
        |      - rng: Function that returns a non-negative integer at random.
        |       Example (Dughmi et al. 2017): if 'rng' is Poisson(lamda) we have
        |       an "exponentiation" Bernoulli factory as follows:
        |       B(p) => B(exp(p*lamda-lamda))
        |
        |  product(self, f1, f2)
        |      Product (conjunction; AND): B(p), B(q) => B(p*q)  (Flajolet et al. 2010)
        |      - f1, f2: Functions that return 1 if heads and 0 if tails.
        |
        |  randbit(self)
        |      Generates a random bit that is 1 or 0 with equal probability.
        |
        |  rndintexc(self, maxexc)
        |      Returns a random integer in [0, maxexc).
        |
        |  sin(self, f)
        |      Sine Bernoulli factory: B(p) => B(sin(p)).  Special
        |      case of Algorithm3 of reverse-time martingale paper.
        |
        |  square(self, f1, f2)
        |      Square: B(p) => B(1-p). (Flajolet et al. 2010)
        |      - f1, f2: Functions that return 1 if heads and 0 if tails.
        |
        |  twocoin(self, f1, f2, c1=1, c2=1, beta=1)
        |      Two-coin Bernoulli factory: B(p), B(q) =>
        |              B(c1*p*beta / (beta * (c1*p+c2*q) - (beta - 1)*(c1+c2)))
        |         (Gonçalves et al. 2017, Vats et al. 2020; in Vats et al.,
        |          C1,p1 corresponds to cy and C2,p2 corresponds to cx).
        |          Logistic Bernoulli factory is a special case with q=1, c2=1, beta=1.
        |      - f1, f2: Functions that return 1 if heads and 0 if tails.
        |      - c1, c2: Factors to multiply the probabilities of heads for f1 and f2,
respectively.
        |          - beta: Early rejection parameter ("portkey" two-coin factory).
        |           When beta = 1, the formula simplifies to B(c1*p/(c1*p+c2*q)).
```

```
 |
 |  twofacpower(self, fbase, fexponent)
 |      Bernoulli factory B(p, q) => B(p^q).
 |      Based on algorithm from (Mendo 2019),
 |      but changed to accept a Bernoulli factory
 |      rather than a fixed value for the exponent.
 |      To the best of my knowledge, I am not aware
 |      of any article or paper that presents this particular
 |      Bernoulli factory (before my articles presenting
 |      accurate beta and exponential generators).
 |      - fbase, fexponent: Functions that return 1 if heads and 0 if tails.
 |        The first is the base, the second is the exponent.
 |
 |  zero_or_one(self, px, py)
 |      Returns 1 at probability px/py, 0 otherwise.
 |
 |  zero_or_one_arctan_n_div_n(self, x, y=1)
 |      Generates 1 with probability arctan(x/y)*y/x; 0 otherwise.
 |          x/y must be in [0, 1]. Uses a uniformly-fast special case of
 |      the two-coin Bernoulli factory, rather than the even-parity construction in
 |      Flajolet's paper, which is not uniformly fast.
 |      Reference: Flajolet et al. 2010.
 |
 |  zero_or_one_exp_minus(self, x, y)
 |      Generates 1 with probability exp(-x/y); 0 otherwise.
 |      Reference: Canonne et al. 2020.
 |
 |  zero_or_one_log1p(self, x, y=1)
 |      Generates 1 with probability log(1+x/y); 0 otherwise.
 |      Reference: Flajolet et al. 2010.  Uses a uniformly-fast special case of
 |      the two-coin Bernoulli factory, rather than the even-parity construction in
 |      Flajolet's paper, which is not uniformly fast.
 |
 |  zero_or_one_pi_div_4(self)
 |      Generates 1 with probability pi/4.
 |      Reference: Flajolet et al. 2010.
 |
 |  zero_or_one_power(self, px, py, n)
 |      Generates 1 with probability (px/py)^n (where n can be
 |      positive, negative, or zero); 0 otherwise.
 |
 |  zero_or_one_power_ratio(self, px, py, nx, ny)
 |      Generates 1 with probability (px/py)^(nx/ny) (where nx/ny can be
 |      positive, negative, or zero); 0 otherwise.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class DiceEnterprise(builtins.object)
 |  Implements the Dice Enterprise algorithm for
 |  turning loaded dice with unknown bias into loaded dice
 |  with a different bias.  Currently, only the case of biased coins
 |  to loaded dice (the Bernoulli Factory problem) is fully
 |  implemented.
 |
 |  Reference: Morina, G., Łatuszyński, K., et al., "From the
```

```
|  Bernoulli Factory to a Dice Enterprise via Perfect
|  Sampling of Markov Chains", arXiv:1912.09229v1 [math.PR], 2019.
|
|  Example:
|
|  >>> ent=DiceEnterprise()
|  >>> # Example 3 from the paper
|  >>> ent.append_poly(1,[[math.sqrt(2),3]])
|  >>> ent.append_poly(0,[[-5,3],[11,2],[-9,1],[3,0]])
|  >>> coin=lambda: 1 if random.random() < 0.60 else 0
|  >>> print([ent.next(coin) for i in range(100)])
|
|  Methods defined here:
|
|  __init__(self)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  append_poly(self, result, poly)
|      Appends a probability in the form of a polynomial.
|      result - A number indicating the result (die roll or coin
|        flip) that will be
|        returned with the probability represented by this polynomial.
|      poly - Polynomial expressed as a list of terms as follows:
|        Each term is a list of two or three items that express one of
|        the polynomial's terms; the first item is the coefficient,
|        the second is the power of p, and the third (optional) is the power
|        of 1-minus p.  Specifically, the term has the following form:
|                term[0] * p**term[1] * (1-p)**term[2].
|        For example, [3, 4, 5] becomes:
|                3 * p**4 * (1-p)**5
|        For best results, the coefficient should be a rational number
|        (such as int or Python's Fraction).
|      Returns this object.
|
|  augment(self)
|      Augments the degree of the function represented
|      by this object, which can improve performance in some cases
|      (for details, see the paper).
|      Returns this object.
|
|  next(self, coin)
|      Returns the next result of the flip from a coin
|      that is transformed from the given coin by the function
|      represented by this Dice Enterprise object.
|      coin - Function that returns 1 (heads) or 0 (tails).
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)

FILE
    /home/rooster/Documents/SharpDevelopProjects/peteroupc.github.io/bernoulli.py


Help on module interval:

NAME
    interval
```

```
DESCRIPTION
    #  Implements interval numbers and interval arithmetic, backed
    #  by Decimal values.
    #
    #  Written by Peter O. Any copyright to this file is released to the Public Domain.
    #  In case this is not possible, this file is also licensed under Creative Commons Zero
    #  (https://creativecommons.org/publicdomain/zero/1.0/).
    #

CLASSES
    builtins.object
        Interval

    class Interval(builtins.object)
     |  An interval of two Decimal values.
     |
     |  Methods defined here:
     |
     |  __add__(self, v)
     |
     |  __max__(a, b)
     |
     |  __min__(a, b)
     |
     |  __mul__(self, v)
     |
     |  __neg__(self)
     |
     |  __pow__(self, v)
     |
     |  __radd__(self, v)
     |
     |  __repr__(self)
     |      Return repr(self).
     |
     |  __rmul__(self, v)
     |
     |  __rsub__(self, v)
     |
     |  __rtruediv__(self, v)
     |
     |  __sub__(self, v)
     |
     |  __truediv__(self, v)
     |
     |  abs(self)
     |
     |  ceil(self)
     |
     |  clamp(self, a, b)
     |
     |  clampleft(self, a)
     |
     |  cos(self)
     |
     |  exp(self)
     |
     |  floor(self)
     |
     |  isAccurateTo(self, v)
     |
```

```
 |  log(self)
 |
 |  magnitude(self)
 |
 |  mignitude(self)
 |
 |  pi(prec=56)
 |
 |  pow(self, v)
 |
 |  rem(self, v)
 |
 |  sin(self)
 |
 |  sqrt(self)
 |
 |  tan(self)
 |
 |  width(self)
 |      NOTE: Not rigorous!
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(cl, v, sup=None, prec=None)
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

FILE
    /home/rooster/Documents/SharpDevelopProjects/peteroupc.github.io/interval.py


Help on module moore:

NAME
    moore

DESCRIPTION
    #  Implements the Moore Rejection Sampler.
    #
    #  Written by Peter O. Any copyright to this file is released to the Public Domain.
    #  In case this is not possible, this file is also licensed under Creative Commons Zero
    #  (https://creativecommons.org/publicdomain/zero/1.0/).
    #

CLASSES
    builtins.object
        MooreSampler

    class MooreSampler(builtins.object)
     |  Moore rejection sampler, for generating independent samples
     |  from continuous distributions in a way that minimizes error,
     |  if the distribution's PDF (probability density function)
     |  uses "well-defined" arithmetic expressions.
     |  It can sample from one-dimensional or multidimensional
```

```
| distributions.  It can also sample from so-called "transdimensional
| distributions" if the distribution is the union of several component
| distributions that may have different dimensions and are associated
| with one of several _labels_.
|
| Parameters:
|
| - pdf: A function that specifies the PDF.  It takes a single parameter that
|     differs as follows, depending on the case:
|       - One-dimensional case: A single Interval. (An Interval is a mathematical
|         object that specifies upper and lower bounds of a number.)
|       - Multidimensional case: A list of Intervals, one for each dimension.
|       - Transdimensional case (numLabels > 1): A list of two items: the Interval
|         or Intervals, followed by a label number (an integer in [0, numLabels)).
|     This function returns an Interval.  For best results,
|     the function should use interval arithmetic throughout.  The area under
|     the PDF need not equal 1 (this sampler works even if the PDF is only known
|     up to a normalizing constant).
| - mn, mx: Specifies the sampling domain of the PDF.  There are three cases:
|       - One-dimensional case: Both mn and mx are numbers giving the domain,
|         which in this case is [mn, mx].
|       - Multidimensional case: Both mn and mx are lists giving the minimum
|         and maximum bounds for each dimension in the sampling domain.
|         In this case, both lists must have the same size.
|       - Transdimensional case: Currently, this class assumes the component
|         distributions share the same sampling domain, which
|         is given depending on the preceding two cases.
|     For this sampler to work, the PDF must be "locally Lipschitz" in the
|     sampling domain, meaning that the function is continuous everywhere
|     in the domain, and has no slope that tends to a vertical slope anywhere in
|     that domain.
| - numlabels: The number of labels associated with the distribution, if it's a
|     transdimensional distribution.  Optional; the default is 1.
| - bitAccuracy: Bit accuracy of the sampler; the sampler will sample from
|     a distribution (truncated to the sampling domain) that is close to the
|     ideal distribution by 2^-bitAccuracy.  The default is 53.
|
| Reference:
| Sainudiin, Raazesh, and Thomas L. York. "An Auto-Validating, Trans-Dimensional,
| Universal Rejection Sampler for Locally Lipschitz Arithmetical Expressions."
| Reliable Computing 18 (2013): 15-54.
|
| The following reference describes an optimization, not yet implemented here:
| Sainudiin, R., 2014. An Auto-validating Rejection Sampler for Differentiable
| Arithmetical Expressions: Posterior Sampling of Phylogenetic Quartets. In
| Constraint Programming and Decision Making (pp. 143-152). Springer, Cham.
|
| Methods defined here:
|
| __init__(self, pdf, mn, mx, numLabels=1, bitAccuracy=53)
|     Initialize self.  See help(type(self)) for accurate signature.
|
| acceptRate(self)
|
| sample(self)
|     Samples a number or vector (depending on the number of dimensions)
|     from the distribution and returns that sample.
|     If the sampler is transdimensional (the number of labels is greater than 1),
|     instead returns a list containing the sample and a random label in the
|     interval [0, numLabels), in that order.
|
| ----------------------------------------------------------------
```

```
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

FILE
    /home/rooster/Documents/SharpDevelopProjects/peteroupc.github.io/moore.py
```