stats_arrays Documentation

Release 0.4.1

Chris Mutel

Aug 24, 2018

Contents

The `stats_arrays` package provides a standard NumPy array interface for defining uncertain parameters used in models, and classes for Monte Carlo sampling. It also plays well with others.

CHAPTER 1

Motivation

- Want a consistent interface to SciPy and NumPy statistical function
- Want to be able to quickly load and save many parameter uncertainty distribution definitions in a portable format
- Want to manipulate and switch parameter uncertainty distributions and variables
- Want simple Monte Carlo random number generators that return a vector of parameter values to be fed into uncertainty or sensitivity analysis
- Want something simple, extensible, documented and tested

The `stats_arrays` package was originally developed for the Brightway2 life cycle assessment framework, but can be applied to any stochastic model.

CHAPTER 2

Example

```
>>> from stats_arrays import *
>>> my_variables = UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 0.5, 'uncertainty_type': NormalUncertainty.id},
...     {'loc': 1.5, 'minimum': 0, 'maximum': 10, 'uncertainty_type':␣
↪TriangularUncertainty.id}
... )
>>> my_variables
array([(2.0, 0.5, nan, nan, nan, False, 3),
       (1.5, nan, nan, 0.0, 10.0, False, 5)],
    dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
           ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
           ('uncertainty_type', 'u1')])
>>> my_rng = MCRandomNumberGenerator(my_variables)
>>> my_rng.next()
array([ 2.74414022,  3.54748507])
>>> # can also be used as an interator
>>> zip(my_rng, xrange(10))
[(array([ 2.96893108,  2.90654471]), 0),
 (array([ 2.31190619,  1.49471845]), 1),
 (array([ 3.02026168,  3.33696367]), 2),
 (array([ 2.04775418,  3.68356226]), 3),
 (array([ 2.61976694,  7.0149952 ]), 4),
 (array([ 1.79914025,  6.55264372]), 5),
 (array([ 2.2389968 ,  1.11165296]), 6),
 (array([ 1.69236527,  3.24463981]), 7),
 (array([ 1.77750176,  1.90119991]), 8),
 (array([ 2.32664152,  0.84490754]), 9)]
```

See a more complete notebook example.

CHAPTER 3

Parameter array

The core data structure for `stats_arrays` is a parameter array, which is made from a special kind of NumPy array called a NumPy structured array which has the following data type:

```
import numpy as np
base_dtype = [
    ('loc', np.float64),
    ('scale', np.float64),
    ('shape', np.float64),
    ('minimum', np.float64),
    ('maximum', np.float64),
    ('negative', np.bool)
]
```

Note: Read more on NumPy data types.

Note: The *negative* column is used for uncertain parameters whose distributions are normally always positive, such as the lognormal, but in this case have negative values.

In general, most uncertainty distributions can be defined by three variables, commonly called *location*, *scale*, and *shape*. The *minimum* and *maximum* values make distributions **bounded**, so that one can, for example, define a normal uncertainty which is always positive.

Warning: Bounds are not applied in the following methods: 1) Distribution functions (`PDF`, `CDF`, etc.) where you supply the input vector. 2) `.statistics`, which gives 95 percent confidence intervals for the unbounded distribution.

3.1 Heterogeneous parameter array

Parameter arrays can have multiple uncertainty distributions. To distinguish between the different distributions, another column, called `uncertainty_type`, is added:

```
heterogeneous_dtype = [
    ('uncertainty_type', np.uint8),
    ('loc', np.float64),
    ('scale', np.float64),
    ('shape', np.float64),
    ('minimum', np.float64),
    ('maximum', np.float64),
    ('negative', np.bool)
]
```

Note that *stats_arrays* was developed in conjunction with the Brightway LCA framework; Brightway uses the field name "uncertainty type", without the underscore. Be sure to use the underscore when using *stats_arrays*.

Each uncertainty distribution has an integer ID number. See the table below for built-in distribution IDs.

Note: The recommended way to use uncertainty distribution IDs is not by looking up the integers manually, but by referring to `SomeClass.id`, e.g. `LognormalDistribution.id`.

3.2 Mapping parameter array columns to uncertainty distributions

Name	ID	`loc`	`scale`	`shape`	`minimum`	`maximum`
Undefined	0	**static value**				
No uncertainty	1	**static value**				
Lognormal[1]	2	μ	σ		*lower bound*	*upper bound*
Normal[2]	3	μ	σ		*lower bound*	*upper bound*
Uniform[3]	4				*minimum*[4]	**maximum**
Triangular[5]	5	*mode*[6]			*minimum*[7]	**maximum**
Bernoulli[8]	6	**p**			*lower bound*	*upper bound*
Discrete Uniform[9]	7				*minimum*[10]	**upper bound**[11]
Weibull[12]	8	*offset*[13]	λ	k		
Gamma[14]	9	*offset*[15]	θ	k		
Beta[16]	10	α	*upper bound*	β		
Generalized Extreme Value[17]	11	μ	σ	ξ		
Student's T[18]	12	*median*	*scale*	ν		

Items in **bold** are required, items in *italics* are optional.

Unused columns can be given any value, but it is recommended that they are set to `np.NaN`.

Warning: Unused optional columns **must** be set to `np.NaN` to avoid unexpected behaviour!

3.3 Extending parameter arrays

Parameter arrays can have additional columns. For example, model parameters that will be inserted into a matrix could have columns called *row* and *column*. For speed reasons, it is recommended that only NumPy numeric types are used if the arrays are to stored on disk.

[1] Lognormal distribution. μ and σ are the mean and standard deviation of the underlying normal distribution
[2] Normal distribution
[3] Uniform distribution
[4] Default is 0 if not otherwise specified
[5] Triangular distribution
[6] Default is $(minimum + maximum)/2$
[7] Default is 0 if not otherwise specified
[8] Bernoulli distribution. If `minimum` **and** `maximum` are specified, p is not limited to $0 < p < 1$, but instead to the interval $(minimum, maximum)$
[9] Discrete uniform
[10] The discrete uniform operates on a "half-open" interval, $[minimum, maximum)$, where the minimum is included but the maximum is not. Default is 0 if not otherwise specified.
[11] The distribution includes values up to, but not including, the `maximum`.
[12] Weibull distribution
[13] Optional offset from the origin
[14] Gamma distribution
[15] Optional offset from the origin
[16] Beta distribution
[17] Extreme value distribution
[18] Student's T distribution

CHAPTER 4

Technical reference

4.1 Probability distributions

4.1.1 UncertaintyBase

class stats_arrays.**UncertaintyBase**

Bases: object

Abstract base class for uncertainty types.

All methods on uncertainty classes should be class methods, as instantiating uncertainty classes many times is not desired.

Defaults

- default_number_points_in_pdf: 200. The default number of points to calculate for PDF/CDF functions.
- standard_deviations_in_default_range: 3. The number of standard deviations that define the default range when calculating PDF/CDF values. In a normal distribution, 3 standard deviations is approximately 99% of all values.

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=50*)

Generate random variables repeatedly until all varaibles are within the bounds of each distribution. Raise MaximumIterationsError if this takes more that *maximum_iterations*. Uses *random_variables* for random number generation.

Inputs

- params : A *Parameter array*.

- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional. Random seed to get repeatable samples.
- maximum_iterations : Integer. Optional. Maximum iterations to try to fit the given bounds before an error is raised.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod cdf(*params*, *vector*)
Used when a distribution is bounded, to determine where to begin or end the percentages used in calculating hypercube sampling space.

Inputs

- params : A *Parameter array*.
- vector : A array of values taken from the uncertainty distributions, with **one row** or the **same number** of rows as *params*.

Output

An array of cumulative densities, bounded on (0,1), with *params* rows and *vector* columns.

classmethod check_2d_inputs(*params*, *vector*)
Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)
Test if there is at least a `threshold` percent chance of generating random numbers within the provided bounds.

Doesn't return anything. Raises `stats_arrays.UnreasonableBoundsError` if this condition is not met.

Inputs

- params : A one-row *Parameter array*.
- threshold : A percentage between 0 and 1. The minimum loc of the distribution covered by the bounds before an error is raised.

classmethod from_dicts(**dicts*)
Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod pdf(*params*, **args*, ***kwargs*)

Provide a standard interface to calculate the probability distribution function of a uncertainty distribution. Default is *cls.default_number_points_in_pdf* points between min to max range if bounds are present, or *cls.standard_deviations_in_default_range* standard distributions.

Inputs

- params : A one-row *Parameter array*.
- xs : Optional. A one-dimensional numpy array of input values.

Output

Important: The output format for PDF is different than CDF or PPF.

A tuple of a vactor x values and a vector of y values. Y values are a one-dimensional array of probability densities, bounded on (0,1), with length *xs*, if provided, or *cls.default_number_points_in_pdf*.

classmethod ppf(*params*, *percentages*)
Return percent point function (inverse of CDF, e.g. value in distribution where x percent of the distribution is less than value) for various distributions.

Inputs

- params : A *Parameter array*.
- percentages : An array of percentages, bounded on (0,1). Each row in *percentages* corresponds to a row in *params*.

Output

An array of values within the ranges of each distribtion, with *params* rows and *percentages* columns.

classmethod random_variables(*params*, *size*, *seeded_random=None*)
Generate random variables for the given uncertainty. Should **not check** to ensure that random samples are with the (minimum, maximum bounds). Bounds checking is provided by the *bounded_random_variables* class method.

Inputs

- params : A *Parameter array*.
- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod statistics(*params*, **args*, ***kwargs*)
Build a dictionary of mean, mode, median, and 95% confidence interval upper and lower values.

Inputs

- params : A one-row *Parameter array*.

Output

{'mean': mean value, 'mode': mode value, 'median': median value, 'upper': upper limit value, 'lower': lower limit value}. All values should be floats (not single-element arrays). Parameters that are not defined should be returned *None*, not omitted.

classmethod validate(*params*)

Validate the parameter array for uncertainty distribution.

Validation is distribution specific. The only default check is that `minimum` is less than or equal to `maximum`, and otherwise raises `stats_arrays.ImproperBoundsError`.

Doesn't return anything.

Args: A *Parameter array*.

BoundedUncertaintyBase

class stats_arrays.**BoundedUncertaintyBase**

Bases: `stats_arrays.distributions.base.UncertaintyBase`

An uncertainty distribution where minimum and maximum bounds are required. No bounds checking is required for these distributions, as bounds are integral inputs into the sample space generator.

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=None*)

No bounds checking because the bounds do not exclude any of the distribution.

classmethod cdf(*params*, *vector*)

Used when a distribution is bounded, to determine where to begin or end the percentages used in calculating hypercube sampling space.

Inputs

- params : A *Parameter array*.
- vector : A array of values taken from the uncertainty distributions, with **one row** or the **same number** of rows as *params*.

Output

An array of cumulative densities, bounded on (0,1), with *params* rows and *vector* columns.

classmethod check_2d_inputs(*params*, *vector*)

Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Always true because the bounds do not exclude any of the distribution.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod pdf(*params*, **args*, ***kwargs*)

Provide a standard interface to calculate the probability distribution function of a uncertainty distribution. Default is *cls.default_number_points_in_pdf* points between min to max range if bounds are present, or *cls.standard_deviations_in_default_range* standard distributions.

Inputs

- params : A one-row *Parameter array*.
- xs : Optional. A one-dimensional numpy array of input values.

Output

Important: The output format for PDF is different than CDF or PPF.

A tuple of a vactor x values and a vector of y values. Y values are a one-dimensional array of probability densities, bounded on (0,1), with length *xs*, if provided, or *cls.default_number_points_in_pdf*.

classmethod ppf(*params*, *percentages*)

Return percent point function (inverse of CDF, e.g. value in distribution where x percent of the distribution is less than value) for various distributions.

Inputs

- params : A *Parameter array*.
- percentages : An array of percentages, bounded on (0,1). Each row in *percentages* corresponds to a row in *params*.

Output

An array of values within the ranges of each distribtion, with *params* rows and *percentages* columns.

classmethod random_variables(*params*, *size*, *seeded_random=None*)

Generate random variables for the given uncertainty. Should **not check** to ensure that random samples are with the (minimum, maximum bounds). Bounds checking is provided by the *bounded_random_variables* class method.

Inputs

- params : A *Parameter array*.
- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod rescale(*params*)
Rescale params to a (0,1) interval. Return adjusted_means and scale. Needed because SciPy assumes a (0,1) interval for many distributions.

classmethod statistics(*params*, **args*, ***kwargs*)
Build a dictionary of mean, mode, median, and 95% confidence interval upper and lower values.

Inputs

- params : A one-row *Parameter array*.

Output

{'mean': mean value, 'mode': mode value, 'median': median value, 'upper': upper limit value, 'lower': lower limit value}. All values should be floats (not single-element arrays). Parameters that are not defined should be returned *None*, not omitted.

4.1.2 Lognormal

class stats_arrays.**LognormalUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=50*)
Generate random variables repeatedly until all varaibles are within the bounds of each distribution. Raise MaximumIterationsError if this takes more that *maximum_iterations*. Uses *random_variables* for random number generation.

Inputs

- params : A *Parameter array*.
- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional. Random seed to get repeatable samples.
- maximum_iterations : Integer. Optional. Maximum iterations to try to fit the given bounds before an error is raised.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod check_2d_inputs(*params*, *vector*)
Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Test if there is at least a `threshold` percent chance of generating random numbers within the provided bounds.

Doesn't return anything. Raises `stats_arrays.UnreasonableBoundsError` if this condition is not met.

Inputs

- params : A one-row *Parameter array*.
- threshold : A percentage between 0 and 1. The minimum loc of the distribution covered by the bounds before an error is raised.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod pdf(*params*, **args*, ***kwargs*)
Generate probability distribution function for lognormal distribution.

classmethod validate(*params*)
Custom validation because mean gets log-transformed

4.1.3 Normal

class stats_arrays.**NormalUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=50*)
Generate random variables repeatedly until all varaibles are within the bounds of each distribution. Raise MaximumIterationsError if this takes more that *maximum_iterations*. Uses *random_variables* for random number generation.

Inputs

- params : A *Parameter array*.
- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional. Random seed to get repeatable samples.
- maximum_iterations : Integer. Optional. Maximum iterations to try to fit the given bounds before an error is raised.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod check_2d_inputs(*params*, *vector*)
Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)
Test if there is at least a `threshold` percent chance of generating random numbers within the provided bounds.

Doesn't return anything. Raises `stats_arrays.UnreasonableBoundsError` if this condition is not met.

Inputs

- params : A one-row *Parameter array*.
- threshold : A percentage between 0 and 1. The minimum loc of the distribution covered by the bounds before an error is raised.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
```

(continues on next page)

(continued from previous page)

```
...      (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...      (5, np.NaN, np.NaN, 3, 10, False, 5)
...      )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
       dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
              ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
              ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

4.1.4 Uniform

class stats_arrays.**UniformUncertainty**

Bases: stats_arrays.distributions.base.BoundedUncertaintyBase

Continuous uniform distribution. In SciPy, the uniform distribution is defined from loc to loc+scale.

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=None*)

No bounds checking because the bounds do not exclude any of the distribution.

classmethod check_2d_inputs(*params*, *vector*)

Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Always true because the bounds do not exclude any of the distribution.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...      {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...      {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...      )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
       dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
              ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
              ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...      (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...      (5, np.NaN, np.NaN, 3, 10, False, 5)
...      )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
       dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
              ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
              ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod rescale(*params*)
Rescale params to a (0,1) interval. Return adjusted_means and scale. Needed because SciPy assumes a (0,1) interval for many distributions.

4.1.5 Discrete Uniform

class stats_arrays.**DiscreteUniform**
Bases: `stats_arrays.distributions.base.UncertaintyBase`

The discrete uniform distribution includes all integer values from the `minimum` up to, but excluding the `maximum`.

See https://en.wikipedia.org/wiki/Uniform_distribution_(discrete).

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=50*)
Generate random variables repeatedly until all varaibles are within the bounds of each distribution. Raise MaximumIterationsError if this takes more that *maximum_iterations*. Uses *random_variables* for random number generation.

Inputs

- params : A *Parameter array*.

- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional. Random seed to get repeatable samples.
- maximum_iterations : Integer. Optional. Maximum iterations to try to fit the given bounds before an error is raised.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod check_2d_inputs(*params*, *vector*)
Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)
Test if there is at least a `threshold` percent chance of generating random numbers within the provided bounds.

Doesn't return anything. Raises `stats_arrays.UnreasonableBoundsError` if this condition is not met.

Inputs

- params : A one-row *Parameter array*.
- threshold : A percentage between 0 and 1. The minimum loc of the distribution covered by the bounds before an error is raised.

classmethod from_dicts(**dicts*)
Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)
Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

4.1.6 Triangular

class stats_arrays.**TriangularUncertainty**

Bases: `stats_arrays.distributions.base.BoundedUncertaintyBase`

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=None*)

No bounds checking because the bounds do not exclude any of the distribution.

classmethod check_2d_inputs(*params*, *vector*)

Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Always true because the bounds do not exclude any of the distribution.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
```

(continues on next page)

(continued from previous page)

```
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
       dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
              ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
              ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
       dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
              ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
              ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod rescale(*params*)

Rescale params to a (0,1) interval. Return adjusted_means and scale. Needed because SciPy assumes a (0,1) interval for many distributions.

4.1.7 Bernoulli

class stats_arrays.**BernoulliUncertainty**

Bases: stats_arrays.distributions.base.BoundedUncertaintyBase

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=None*)

No bounds checking because the bounds do not exclude any of the distribution.

classmethod check_2d_inputs(*params*, *vector*)

Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Always true because the bounds do not exclude any of the distribution.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

classmethod pdf(*params*, **args*, ***kwargs*)

Provide a standard interface to calculate the probability distribution function of a uncertainty distribution. Default is *cls.default_number_points_in_pdf* points between min to max range if bounds are present, or *cls.standard_deviations_in_default_range* standard distributions.

Inputs

- params : A one-row *Parameter array*.
- xs : Optional. A one-dimensional numpy array of input values.

Output

Important: The output format for PDF is different than CDF or PPF.

A tuple of a vactor x values and a vector of y values. Y values are a one-dimensional array of probability densities, bounded on (0,1), with length *xs*, if provided, or *cls.default_number_points_in_pdf*.

classmethod rescale(*params*)

Rescale params to a (0,1) interval. Return adjusted_means and scale. Needed because SciPy assumes a (0,1) interval for many distributions.

classmethod statistics(*params*, **args*, ***kwargs*)

Build a dictionary of mean, mode, median, and 95% confidence interval upper and lower values.

Inputs

- params : A one-row *Parameter array*.

Output

{'mean': mean value, 'mode': mode value, 'median': median value, 'upper': upper limit value, 'lower': lower limit value}. All values should be floats (not single-element arrays). Parameters that are not defined should be returned *None*, not omitted.

4.1.8 Beta

class stats_arrays.**BetaUncertainty**

Bases: stats_arrays.distributions.base.UncertaintyBase

The Beta distribution has the probability distribution function:

$$f(x;\alpha,\beta) = \frac{1}{B(\alpha,\beta)}x^{\alpha-1}(1-x)^{\beta-1},$$

where the normalisation, *B*, is the beta function:

$$B(\alpha,\beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt$$

The α parameter is `loc`, and β is `shape`. By default, the Beta distribution is defined from 0 to 1; the upper bound can be rescaled with the `scale` parameter.

Wikipedia: Beta distribution

classmethod bounded_random_variables(*params*, *size*, *seeded_random=None*, *maximum_iterations=50*)

Generate random variables repeatedly until all varaibles are within the bounds of each distribution. Raise MaximumIterationsError if this takes more that *maximum_iterations*. Uses *random_variables* for random number generation.

Inputs

- params : A *Parameter array*.
- size : Integer. The number of values to draw from each distribution in *params*.
- seeded_random : Integer. Optional. Random seed to get repeatable samples.
- maximum_iterations : Integer. Optional. Maximum iterations to try to fit the given bounds before an error is raised.

Output

An array of random values, with dimensions *params* rows by *size*.

classmethod check_2d_inputs(*params*, *vector*)

Convert `vector` to 2 dimensions if not already, and raise `stats_arrays.InvalidParamsError` if `vector` and `params` dimensions don't match.

classmethod check_bounds_reasonableness(*params*, **args*, ***kwargs*)

Test if there is at least a `threshold` percent chance of generating random numbers within the provided bounds.

Doesn't return anything. Raises `stats_arrays.UnreasonableBoundsError` if this condition is not met.

Inputs

- params : A one-row *Parameter array*.
- threshold : A percentage between 0 and 1. The minimum loc of the distribution covered by the bounds before an error is raised.

classmethod from_dicts(**dicts*)

Construct a *Heterogeneous parameter array* from parameter dictionaries.

Dictionary keys are the normal parameter array columns. Each distribution defines which columns are required and which are optional.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_dicts(
...     {'loc': 2, 'scale': 3, 'uncertainty_type': 3},
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5}
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more dictionaries.

Returns: A *Heterogeneous parameter array*

classmethod from_tuples(**data*)

Construct a *Heterogeneous parameter array* from parameter tuples.

The order of the parameters is:

1. `loc`
2. `scale`
3. `shape`
4. `minimum`
5. `maximum`
6. `negative`
7. `uncertainty_type`

Each input tuple must have a length of exactly 7. For more flexibility, use `from_dicts`.

Example:

```
>>> from stats_arrays import UncertaintyBase
>>> import numpy as np
>>> UncertaintyBase.from_tuples(
...     (2, 3, np.NaN, np.NaN, np.NaN, False, 3),
...     (5, np.NaN, np.NaN, 3, 10, False, 5)
...     )
array([(2.0, 3.0, nan, nan, nan, False, 3),
       (5.0, nan, nan, 3.0, 10.0, False, 5)],
      dtype=[('loc', '<f8'), ('scale', '<f8'), ('shape', '<f8'),
             ('minimum', '<f8'), ('maximum', '<f8'), ('negative', '?'),
             ('uncertainty_type', 'u1')])
```

Args: One of more tuples of length 7.

Returns: A *Heterogeneous parameter array*

4.1.9 Generalized Extreme Value

class stats_arrays.**GeneralizedExtremeValueUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

The generalized extreme value uncertainty, or Fisher-Tippett, distribution is described in the Wikipedia article: http://en.wikipedia.org/wiki/Generalized_extreme_value_distribution.

In our implementation, μ is `location`, σ is `scale`, and ξ is `shape`.

4.1.10 Student's T

class stats_arrays.**StudentsTUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

The Student's T uncertainty distribution probability density function is a function of ν, the degrees of freedom:

$$f(x;\nu) = \frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi},\Gamma(\frac{\nu}{2})}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

A non-standardized distribution, with a location and scale parameter, is also possible, through the transformation:

$$X = \mu + \sigma f$$

In our implementation, the location parameter μ is `location`, the scale parameter σ is `scale`, and ν (the degrees of freedom) is `shape`.

See http://en.wikipedia.org/wiki/Student%27s_t-distribution

4.1.11 Gamma

class stats_arrays.**GammaUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

The Gamma uncertainty distribution probability density function as a function of k, the shape parameters, and θ, the scale parameter:

$$f(x;k,\theta) = \frac{x^{k-1}e^{-\frac{x}{\theta}}}{\theta^k\Gamma(k)}$$

The scale parameter k is `shape`, and θ is `scale`. An optional location parameter, which offsets the distribution from the origin, can be specified in `loc`.

See https://en.wikipedia.org/wiki/Gamma_distribution.

4.1.12 Weibull

class stats_arrays.**WeibullUncertainty**
Bases: stats_arrays.distributions.base.UncertaintyBase

The Weibull distribution has the probability distribution function:

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1}e^{-(x/\lambda)^k}$$

In our implementation, λ is `scale`, and k is `shape`. An optional location parameter, which offsets the distribution from the origin, can be specified in `loc`.

See https://en.wikipedia.org/wiki/Weibull_distribution.

4.2 Random number generators

4.2.1 Random number generator

class stats_arrays.**RandomNumberGenerator**(*uncertainty_type*, *params*, *size=1*, *maximum_iterations=100*, *seed=None*, ***kwargs*)

__init__(*uncertainty_type*, *params*, *size=1*, *maximum_iterations=100*, *seed=None*, ***kwargs*)

Create a random number generator from a *Parameter array* and an uncertainty distribution.

Upon instantiation, the class checks that:

- The `minimum` and `maximum` bounds, if any, are reasonable
- The given uncertainty type can be used

`uncertainty_type` is not required to be a subclass of *UncertaintyBase*, but needs to have the method `bounded_random_variables`.

The returned class instance can be called directly:

```
>>> from stats_arrays import RandomNumberGenerator, TriangularUncertainty
>>> params = TriangularUncertainty.from_dicts(
...     {'loc': 5, 'minimum': 3, 'maximum': 10},
...     {'loc': 1, 'minimum': 0.7, 'maximum': 4.4}
...     )
>>> rng = RandomNumberGenerator(TriangularUncertainty, params)
>>> rng.generate_random_numbers()
array([[ 8.00843856],
   [ 1.54968237]])
```

but can also be used as an iterator:

```
>>> zip(range(2), rng)
[(0, array([[ 5.34298156],
   [ 1.02447677]])),
 (1, array([[ 5.45360508],
   [ 1.99372889]]))]
```

Args:

- **uncertainty_type** (object): An uncertainty type class (subclass of `stats_arrays.distributions.UncertaintyBase`)
- **params** (array): The *Parameter array*
- *size* (int, optional): The number of samples to draw from each parameter. Default is `1`.
- *maximum_iterations* (int, optional): The number of times to draw samples that fit within the given bounds, if any, before raising `stats_arrays.MaximumIterationsError`. Default is `100`.
- *seed* (int, optional): Seed value for the random number generator. Default is `None`.

Returns: A class instance

verify_params(*params=None*, *uncertainty_type=None*)

Verify that parameters are within bounds. Mean is not restricted to bounds, unless the distribution requires it (e.g. triangular).

verify_uncertainty_type(*uncertainty_type=None*)
Make sure the given uncertainty type provides the method `bounded_random_variables`.

4.2.2 Monte Carlo random number generator

class stats_arrays.**MCRandomNumberGenerator**(*params*, *maximum_iterations=50*, *seed=None*, ***kwargs*)
A Monte Carlo random number generator that operates on a *Heterogeneous parameter array*.

Upon instantiation, the class checks that:

- Each unique `uncertainty_type` is a valid choice in `uncertainty_choices`
- That the parameter array for each uncertainty type validates

The returned class instance can be called directly with `next`, or can be used as an iterator:

```
>>> from stats_arrays import MCRandomNumberGenerator, UncertaintyBase
>>> params = UncertaintyBase.from_dicts(
...     {'loc': 5, 'minimum': 3, 'maximum': 10, 'uncertainty_type': 5},
...     {'loc': 1, 'scale': 0.7, 'uncertainty_type': 3}
...     )
>>> mcrng = MCRandomNumberGenerator(params)
>>> zip(range(2), mcrng)
[(0, array([ 1.35034874,  5.2705415 ])),
 (1, array([ 5.2705415 ,  1.35034874]))]
```

Args:

- **params** (array): The *Heterogeneous parameter array*
- *maximum_iterations* (int, optional): The number of times to draw samples that fit within the given bounds, if any, before raising `stats_arrays.MaximumIterationsError`. Default is `100`.
- *seed* (int, optional): Seed value for the random number generator. Default is `None`.

Returns: A class instance

__init__(*params*, *maximum_iterations=50*, *seed=None*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

get_positions()
Construct dictionary of where each distribution starts and stops in the sorted parameter array

next()
Generate a new vector of random numbers

verify_params()
Verify that all uncertainty types are allowed, and parameter validate using distribution class methods

4.2.3 Latin Hypercubic sampling

class stats_arrays.**LatinHypercubeRNG**(*params*, *seed=None*, *samples=10*, ***kwargs*)
A random number generator that pre-calculates a sample space to draw from.

Inputs

- params : A *Parameter array* which gives parameters for distributions (one distribution per row).
- seed : An integer (or array of integers) to seed the NumPy random number generator.
- samples : An integer number of samples to construct for each distribution.

__init__(*params*, *seed=None*, *samples=10*, ***kwargs*)
x.__init__(...) initializes x; see help(type(x)) for signature

build_hypercube()
Build an array, of shape *self.length* rows by *self.samples* columns, which contains the sample space to be drawn from when doing Latin Hypercubic sampling.

Each row represents a different data point and distribution. The final sample space is *self.hypercube*. All distributions from *uncertainty_choices* are usable, and bounded distributions are also fine.

Builds

self.hypercube : Numpy array with dimensions *self.length* by *self.samples*.

next()
Draw directly from pre-computed sample space.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Index

Symbols

B

C

D

F

G

L

M

N

P

R

S

T

U

V

W