steppyngstounes: Iterators for Python

Jonathan E. Guyer

Dec 05, 2023

Materials Science and Engineering Division
Material Measurement Laboratory
National Institute of Standards and Technology

Contents

---

steppyngstounes / ˈstɛp ɪŋˌstoʊnz /

1. *pl. n. [Middle English]* Stones used as steps of a stairway; also, stones in a stream used for crossing.[1]

   …while at Calais in 1474 we find 40 'steppyngstounes' bought for the stairways of the town.[2]

2. *n. [chiefly Pythonic]* A package that provides iterators for advancing from *start* to *stop*, subject to algorithms that depend on user-defined *value* or *error*.

[1] *Middle English Dictionary*, Ed. Robert E. Lewis, *et al.*, Ann Arbor: University of Michigan Press, 1952-2001. Online edition in *Middle English Compendium*, Ed. Frances McSparran, *et al.*, Ann Arbor: University of Michigan Library, 2000-2018. <https://quod.lib.umich.edu/m/middle-english-dictionary/dictionary/MED42815>. Accessed 16 December 2020.

[2] *Building in England, Down to 1540: A Documentary History*, L. F. Salzman, Clarenden Press, Oxford, 1952. <https://books.google.com/books?id=WtZPAAAAMAAJ&focus=searchwithinvolume&q=steppyngstounes>. Accessed 16 December 2020.

Testing and Coverage failing (https://github.com/usnistgov/steppyngstounes/actions/workflows/testing-and-coverage.yml) (https://github.com/usnistgov/steppyngstounes/actions/workflows/build-docs.yml) Linting and Spelling passing (https://github.com/usnistgov/steppyngstounes/actions/workflows/linting-and-spelling.yml) contributors 6 (https://github.com/guyer/steppyngstounes) code quality A (https://www.codacy.com/gh/guyer/steppyngstounes/dashboard?utm_source=github.com&utm_medium=referral&utm_content=

Computations that evolve in time or sweep a variable often boil down to a control loop like

```
for step in range(steps):
    do_something(step)
```

or

```
t = 0
while t < totaltime:
    t += dt
    do_something(dt)
```

which works well enough, until the size of the steps needs to change. This can be to save or plot results at some fixed points, or because the computation becomes either harder or easier to perform. The control loop then starts to dominate the script, obscuring the interesting parts of the computation, particularly as different edge cases are accounted for.

Packages like odeint (https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html) address many of these issues, but do so through callback functions, which effectively turn the computation of interest inside out, again obscuring the interesting bits. Further, because they are often tailored for applications like solving ordinary differential equations, applying them to other stepping problems, even solving partial differential equations (https://www.ctcms.nist.gov/fipy), can be rather opaque.

The steppyngstounes package is designed to retain the simplicity of the original control loop, while allowing great flexibility in how steps are taken and automating all of the aspects of increasing and decreasing the step size.

A steppyngstounes control loop can be as simple as

```
from steppyngstounes import FixedStepper

for step in FixedStepper(start=0., stop=totaltime, size=dt):
    do_something(step.size)

    _ = step.succeeded()
```

which replicates the `while` construct above, but further ensures that `totaltime` is not overshot if it isn't evenly divisible by `dt`.

> **Attention:** The call to *`succeeded()`* informs the *`Stepper`* to advance, otherwise it will iterate on the same step indefinitely.

Rather than manually incrementing the control variable (e.g., `t`), the values of the control variable before and after the step are available as the *`Step`* attributes `begin` and `end`. The attribute `size` is a shorthand for `step.end - step.begin`.

If the size of the steps should be adjusted by some characteristic of the calculation, such as the change in the value since the last solution, the error (normalized to 1) can be passed to *`succeeded()`*, causing the *`Stepper`* to advance (possibly adjusting the next step size) or to retry the step with a smaller step size.

```python
from steppyngstounes import SomeStepper

old = initial_condition
for step in SomeStepper(start=0., stop=totaltime, size=dt):
    new = do_something_else(step.begin, step.end, step.size)

    err = (new - old) / scale

    if step.succeeded(error=err):
        old = new
        # do happy things
    else:
        # do sad things
```

A hierarchy of `Stepper` iterations enables saving or plotting results at fixed, possibly irregular, points, while allowing an adaptive `Stepper` to find the most efficient path between those checkpoints.

```python
from steppyngstounes import CheckpointStepper, SomeStepper

old = initial_condition
for checkpoint in CheckpointStepper(start=0.,
                                    stops=[1e-3, 1, 1e3, 1e6]):

    for step in SomeStepper(start=checkpoint.begin,
                            stop=checkpoint.end,
                            size=checkpoint.size):

        new = do_something_else(step.begin, step.end, step.size)

        err = (new - old) / scale

        if step.succeeded(error=err):
            old = new
            # do happy things
        else:
            # do sad things

    save_or_plot()

    _ = checkpoint.succeeded()
```

A variety of stepping algorithms are described and demonstrated in the documentation of the individual `steppyngstounes` classes.

---

1 Dependencies

1.1 Using

- numpy (https://numpy.org/)
- scipy (https://scipy.org/)

1.2 Testing

- pytest (https://pytest.org/)

1.3 Documenting

- sphinx (https://www.sphinx-doc.org/) >= 3.1
- matplotlib (https://matplotlib.org/)

2 Installing

```
$ python setup.py install
```

3 Testing

```
$ pytest
```

4 Building the Documentation

```
$ python setup.py build_sphinx
```

If the figures do not update

```
$ touch docs/_autosummary/*.rst
```

and repeat.

If the documentation seems not to build correctly in other respects:

```
$ python setup.py build_sphinx --all-files --fresh-env
```

Documentation can be found in `STEPPYNGSTOUNES/build/sphinx/html`.

5 Support

For help using this package, file an issue (https://github.com/guyer/steppyngstounes/issues) on the GitHub repository (https://github.com/guyer/steppyngstounes). Contributions are welcome via pull request (https://github.com/guyer/steppyngstounes/pulls).

6 API

Description of specific *Stepper* classes:

steppyngstounes

6.1 steppyngstounes

Modules

steppyngstounes.checkpointStepper
steppyngstounes.fixedStepper
steppyngstounes.parsimoniousStepper
steppyngstounes.pidStepper
steppyngstounes.pseudoRKQSStepper
steppyngstounes.scaledStepper
steppyngstounes.sequenceStepper
steppyngstounes.stepper

steppyngstounes.checkpointStepper

Classes

class steppyngstounes.checkpointStepper.**CheckpointStepper**(*start*, *stops*, *stop=inf*, *inclusive=False*, *record=False*)

Bases: *Stepper*

Stepper that stops at fixed points.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stops** (*iterable of float*) – Desired checkpoints.
- **stop** (*float, optional*) – Finish of range to step over (default *np.inf*). In the event that any of *stops* exceed *stop*, the stepper will terminate at *stop*. A step will not be taken to *stop* otherwise (clear?).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)
- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import CheckpointStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_\mathrm{max}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, t_max is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

```
>>> stepper = CheckpointStepper(start=0., stop=totaltime, inclusive=True,
...                             stops=10.**np.arange(-5, 5), record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     _ = step.succeeded(value=new)
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
10 succesful steps in 10 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

As this stepper doesn't use the error, we don't expect the post hoc error to satisfy the tolerance.

property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to `Stepper` via `succeeded()`.

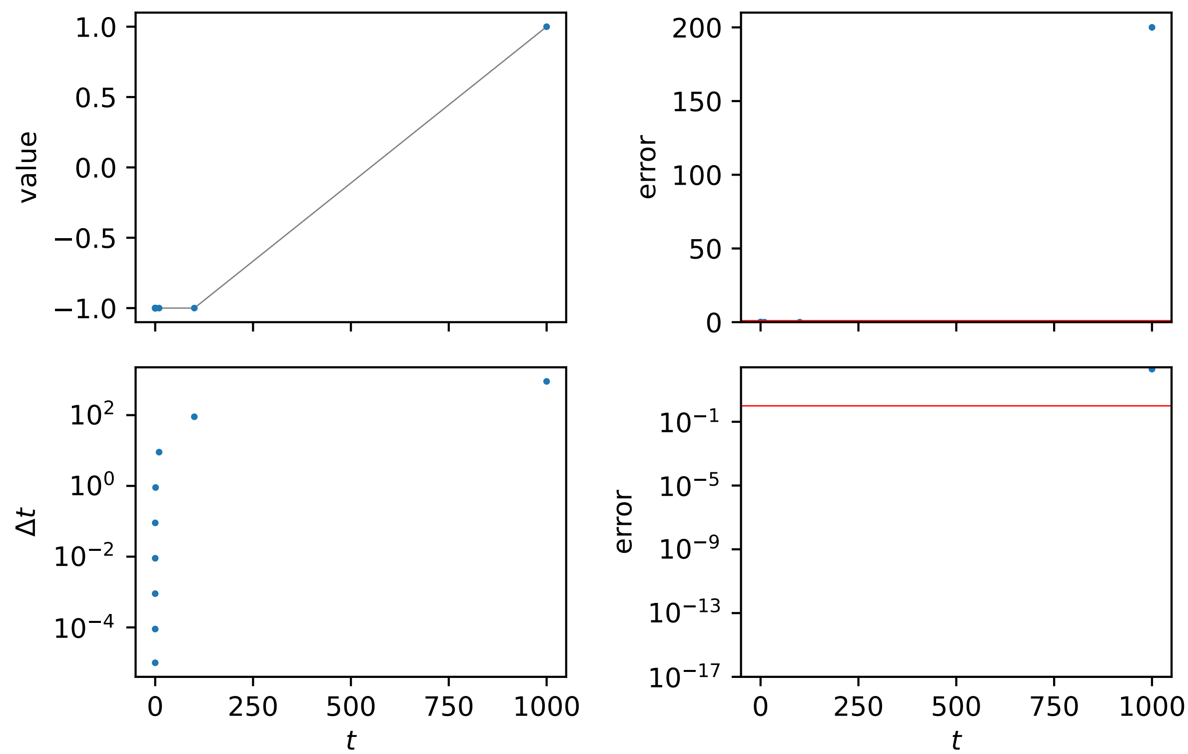
next()

Return the next step.

Note: Legacy Python 2.7 support.

Return type
`Step`

10 successful `CheckpointStepper` steps and trajectory of 10 attempts



Raises

`StopIteration` – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

ndarray of values of the control variable attempted so far.

succeeded (*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns

Whether step was successful.

Return type

bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

steppyngstounes.fixedStepper

Classes

class steppyngstounes.fixedStepper.**FixedStepper** (*start*, *stop*, *size=None*, *minStep=None*, *inclusive=False*, *record=False*, *limiting=False*)

Bases: *Stepper*

Stepper that takes steps of constant size.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **size** (*float*) – Desired step size.
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)

- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import FixedStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\max}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, $t_{\max}$ is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

```
>>> stepper = FixedStepper(start=0., stop=totaltime, inclusive=True,
...                        size=3., record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     _ = step.succeeded(value=new)
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
335 succesful steps in 335 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

As this stepper doesn't use the error, we don't expect the post hoc error to satisfy the tolerance.

property errors

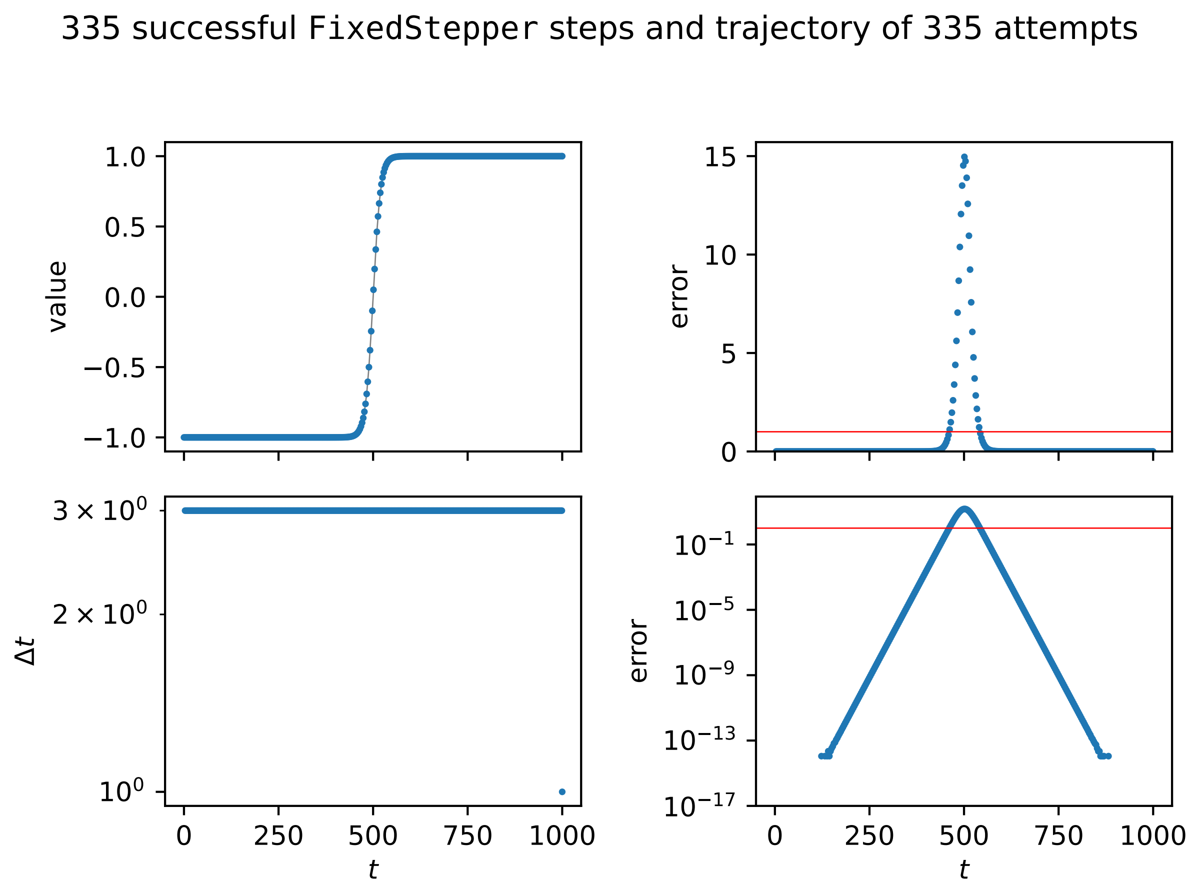
ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to *Stepper* via *succeeded()*.

next()

Return the next step.

Note: Legacy Python 2.7 support.


335 successful FixedStepper steps and trajectory of 335 attempts
value
1.0
0.5
0.0
−0.5
−1.0
error
15
10
5
0
Δt
3 × 10⁰
2 × 10⁰
10⁰
0
250
500
750
1000
t
error
10⁻¹
10⁻⁵
10⁻⁹
10⁻¹³
10⁻¹⁷
0
250
500
750
1000
t

Return type
`Step`

Raises
`StopIteration` – If there are no further steps to take

`property sizes`
ndarray of the step size at each step attempt.

`property steps`
ndarray of values of the control variable attempted so far.

`succeeded`(*step*, *value=None*, *error=None*)
Test if step was successful.

Stores data about the last step.

Parameters

- **`step`** (`Step`) – The step to test.
- **`value`** (`float, optional`) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which `Stepper` is being used. (default None).
- **`error`** (`float, optional`) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which `Stepper` is being used. (default None).

Returns
Whether step was successful.

Return type
bool

`property successes`
ndarray of whether the step was successful at each step attempt.

`property values`
ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to `Stepper` via `succeeded()`.

steppyngstounes.parsimoniousStepper

Classes

`class` `steppyngstounes.parsimoniousStepper.`**`ParsimoniousStepper`**(*start*, *stop*, *N*, *minStep=0.0*, *inclusive=False*, *scale='dl'*, *minsteps=4*, *maxinitial=11*)

Bases: `Stepper`

Non-monotonic stepper that samples sparsely explored regions

Computes the function where the curvature is highest and where not many points have been computed.

Note: By its nature, this `Stepper` must *record*.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **N** (*int*) – Number of points to sample.
- **minStep** (*float*) – Smallest step to allow (default *(stop - start) * eps* (https://numpy.org/doc/stable/reference/generated/numpy.finfo.html)).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)
- **scale** (*str*) – Parameter to indicate whether to scale by value "dy" or arc length "dl" (default "dl").
- **minsteps** (*int*) – Minimum number of steps to take (default 4).
- **maxinitial** (*int*) – The maximum number of even steps to take before adapting (default 11).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import ParsimoniousStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\max}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, $t_{\max}$ is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

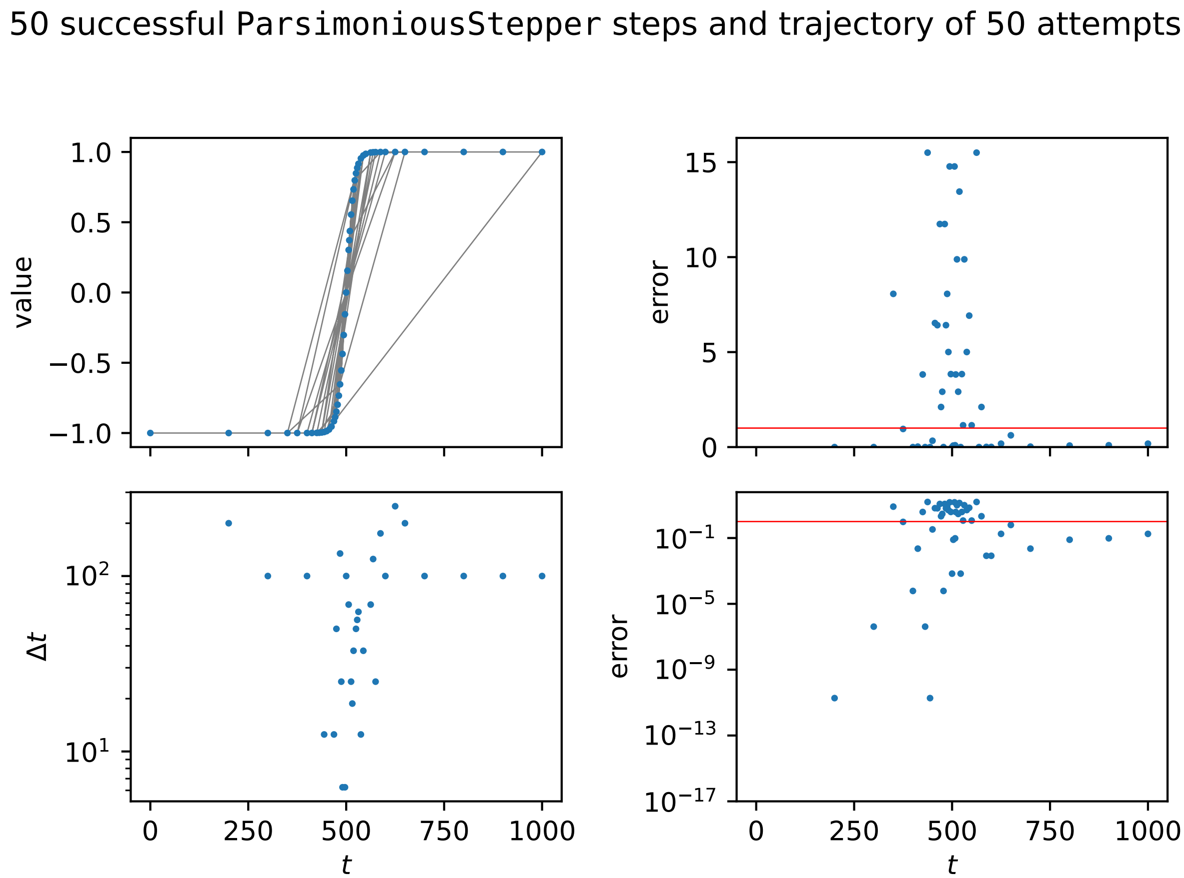
Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

```
>>> stepper = ParsimoniousStepper(start=0., stop=totaltime, inclusive=True,
...                               N=50)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     _ = step.succeeded(value=new)
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
50 succesful steps in 50 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

As this stepper doesn't use the error, we don't expect the post hoc error to satisfy the tolerance.



property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to `Stepper` via `succeeded()`.

next()

Return the next step.

Note: Legacy Python 2.7 support.

Return type
`Step`

Raises
StopIteration – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

ndarray of values of the control variable attempted so far.

succeeded(*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns

Whether step was successful.

Return type

bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

steppyngstounes.pidStepper

Classes

class steppyngstounes.pidStepper.**PIDStepper**(*start*, *stop*, *size=None*, *minStep=None*, *inclusive=False*, *record=False*, *limiting=True*, *proportional=0.075*, *integral=0.175*, *derivative=0.01*)

Bases: *Stepper*

Adaptive stepper using a PID controller.

Calculates a new step as

$$\Delta_{n+1} = \left(\frac{e_{n-1}}{e_n}\right)^{k_P} \left(\frac{1}{e_n}\right)^{k_I} \left(\frac{e_{n-1}^2}{e_n e_{n-2}}\right)^{k_D} \Delta_n$$

where Δ_n is the step size for step n and e_n is the error at step n. k_P is the proportional coefficient, k_I is the integral coefficient, and k_D is the derivative coefficient.

On failure, retries with

$$\Delta_n = \min\left(\frac{1}{e_n}, 0.8\right) \Delta_n$$

Based on:

```
@article{PIDpaper,
   author =  {A. M. P. Valli and G. F. Carey and A. L. G. A. Coutinho},
   title =   {Control strategies for timestep selection in finite
              element simulation of incompressible flows and
              coupled reaction-convection-diffusion processes},
   journal = {Int. J. Numer. Meth. Fluids},
   volume =  47,
   year =    2005,
   pages =   {201-231},
   doi =     {10.1002/fld.805},
}
```

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **size** (*float*) – Suggested step size to try (default None).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)
- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).
- **limiting** (*bool*) – Whether to prevent error from exceeding 1 (default True).
- **minStep** (*float*) – Smallest step to allow (default *(stop - start)* * *eps* (https://numpy.org/doc/stable/reference/generated/numpy.finfo.html)).
- **proportional** (*float*) – PID control k_P coefficient (default 0.075).
- **integral** (*float*) – PID control k_I coefficient (default 0.175).
- **derivative** (*float*) – PID control k_D coefficient (default 0.01).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import PIDStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\text{max}}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, t_{max} is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

```
>>> old = -1.
>>> stepper = PIDStepper(start=0., stop=totaltime, inclusive=True,
...                      record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     error = abs(new - old) / errorscale
...
...     if step.succeeded(value=new, error=error):
...         old = new
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
256 succesful steps in 274 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

Check that the post hoc error satisfies the desired tolerance.

```
>>> print(max(errors) < 1.)
True
```

property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to *Stepper* via *succeeded()*.

next()

Return the next step.

Note: Legacy Python 2.7 support.

Return type
Step

Raises
StopIteration – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

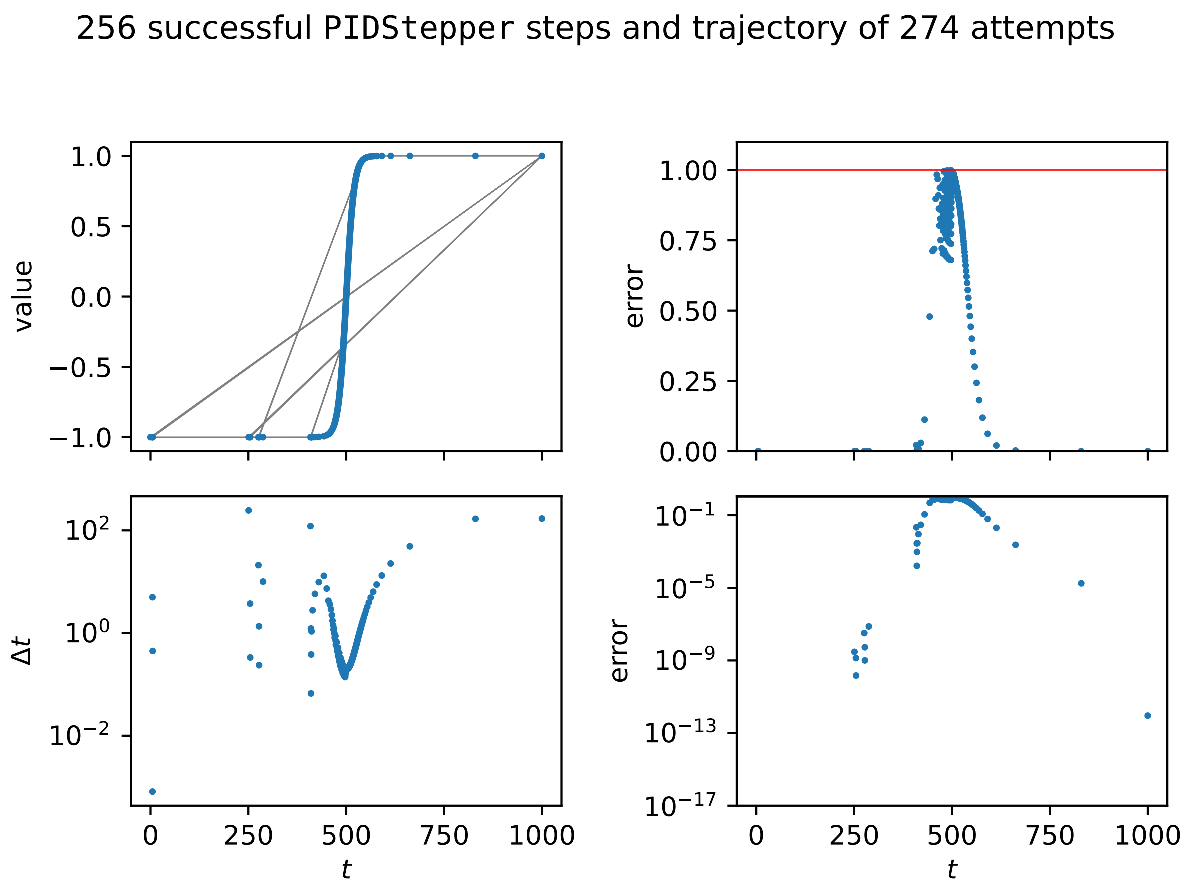
ndarray of values of the control variable attempted so far.

succeeded(*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters


256 successful PIDStepper steps and trajectory of 274 attempts
value
error
Δt
error
t
t
1.0
0.5
0.0
−0.5
−1.0
1.00
0.75
0.50
0.25
0.00
10^2
10^0
10^−2
10^−1
10^−5
10^−9
10^−13
10^−17
0
250
500
750
1000

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns
Whether step was successful.

Return type
bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to `Stepper` via `succeeded()`.

steppyngstounes.pseudoRKQSStepper

Classes

class steppyngstounes.pseudoRKQSStepper.**PseudoRKQSStepper**(*start*, *stop*, *size=None*, *minStep=None*, *inclusive=False*, *record=False*, *limiting=True*, *safety=0.9*, *pgrow=-0.2*, *pshrink=-0.25*, *maxgrow=5*, *minshrink=0.1*)

Bases: `Stepper`

Pseudo-Runge-Kutta adaptive stepper.

Based on the `rkqs` (Runge-Kutta "quality-controlled" stepper) algorithm of Numerical Recipes in C: 2nd Edition, Section 16.2.

Not really appropriate, since we're not doing the *rkck* Runge-Kutta steps in the first place, but works OK.

Calculates a new step as

$$\Delta_{n+1} = \min\left[S\left(e_n\right)^{P_\text{grow}}, f_\text{max}\right]\Delta_n$$

where Δ_n is the step size for step n and e_n is the error at step n. S is the safety factor, P_grow is the growth exponent, and f_max is the maximum factor to grow the step size.

On failure, retries with

$$\Delta_n = \max\left[S\left(e_n\right)^{P_\text{shrink}}, f_\text{min}\right]\Delta_n$$

where P_shrink is the shrinkage exponent and f_min is the minimum factor to shrink the stepsize.

Parameters

- **start** (*float*) – Beginning of range to step over.

- **stop** (*float*) – Finish of range to step over.
- **size** (*float*) – Suggested step size to try (default None).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)
- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).
- **limiting** (*bool*) – Whether to prevent error from exceeding 1 (default True).
- **minStep** (*float*) – Smallest step to allow (default *(stop - start)* * *eps* (https://numpy.org/doc/stable/reference/generated/numpy.finfo.html)).
- **safety** (*float*) – RKQS control safety factor S (default 0.9).
- **pgrow** (*float*) – RKQS control growth exponent P_{grow} (default -0.2).
- **pshrink** (*float*) – RKQS control shrinkage exponent P_{shrink} (default -0.25).
- **maxgrow** (*float*) – RKQS control maximum factor to grow step size f_{max} (default 5).
- **minshrink** (*float*) – RKQS control minimum factor to shrink step size f_{min} (default 0.1).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import PseudoRKQSStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\text{max}}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, t_{max} is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

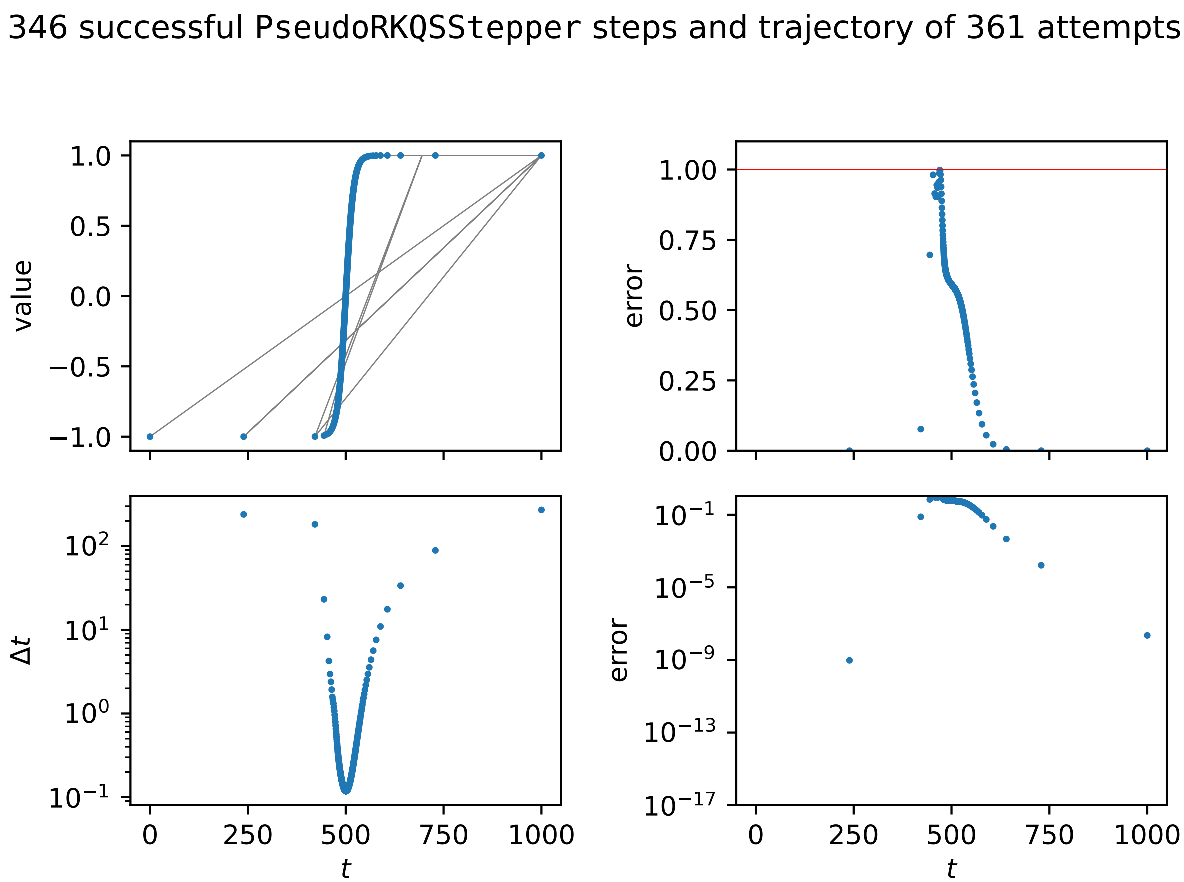
```
>>> old = -1.
>>> stepper = PseudoRKQSStepper(start=0., stop=totaltime, inclusive=True,
...                             record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     error = abs(new - old) / errorscale
...
...     if step.succeeded(value=new, error=error):
...         old = new
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
346 succesful steps in 361 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

Check that the post hoc error satisfies the desired tolerance.

```
>>> print(max(errors) < 1.)
True
```



property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to `Stepper` via `succeeded()`.

next()

Return the next step.

Note: Legacy Python 2.7 support.

Return type

`Step`

Raises
`StopIteration` – If there are no further steps to take

property sizes
ndarray of the step size at each step attempt.

property steps
ndarray of values of the control variable attempted so far.

succeeded (*step*, *value=None*, *error=None*)
Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns
Whether step was successful.

Return type
bool

property successes
ndarray of whether the step was successful at each step attempt.

property values
ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

steppyngstounes.scaledStepper

Classes

class steppyngstounes.scaledStepper.**ScaledStepper** (*start*, *stop*, *size=None*, *minStep=None*, *inclusive=False*, *record=False*, *growFactor=1.2*, *shrinkFactor=0.5*)

Bases: *Stepper*

Adaptive stepper that adjusts the step by fixed factors.

Calculates a new step as

$$\Delta_{n+1} = f_{\text{grow}} \Delta_n$$

where Δ_n is the step size for step n and f_{grow} is the factor by which to grow the step size.

On failure, retries with

$$\Delta_n = f_{\text{shrink}} \Delta_n$$

where f_{shrink} is the factor by which to shrink the step size.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **size** (*float*) – Suggested step size to try (default None).
- **minStep** (*float*) – Smallest step to allow (default *(stop - start)* * *eps* (https://numpy.org/doc/stable/reference/generated/numpy.finfo.html)).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)
- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).
- **growFactor** (*float*) – Growth factor f_{grow} (default 1.2).
- **shrinkFactor** (*float*) – Shrinkage factor f_{shrink} (default 0.5).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import ScaledStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\max}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, $t_{\max}$ is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

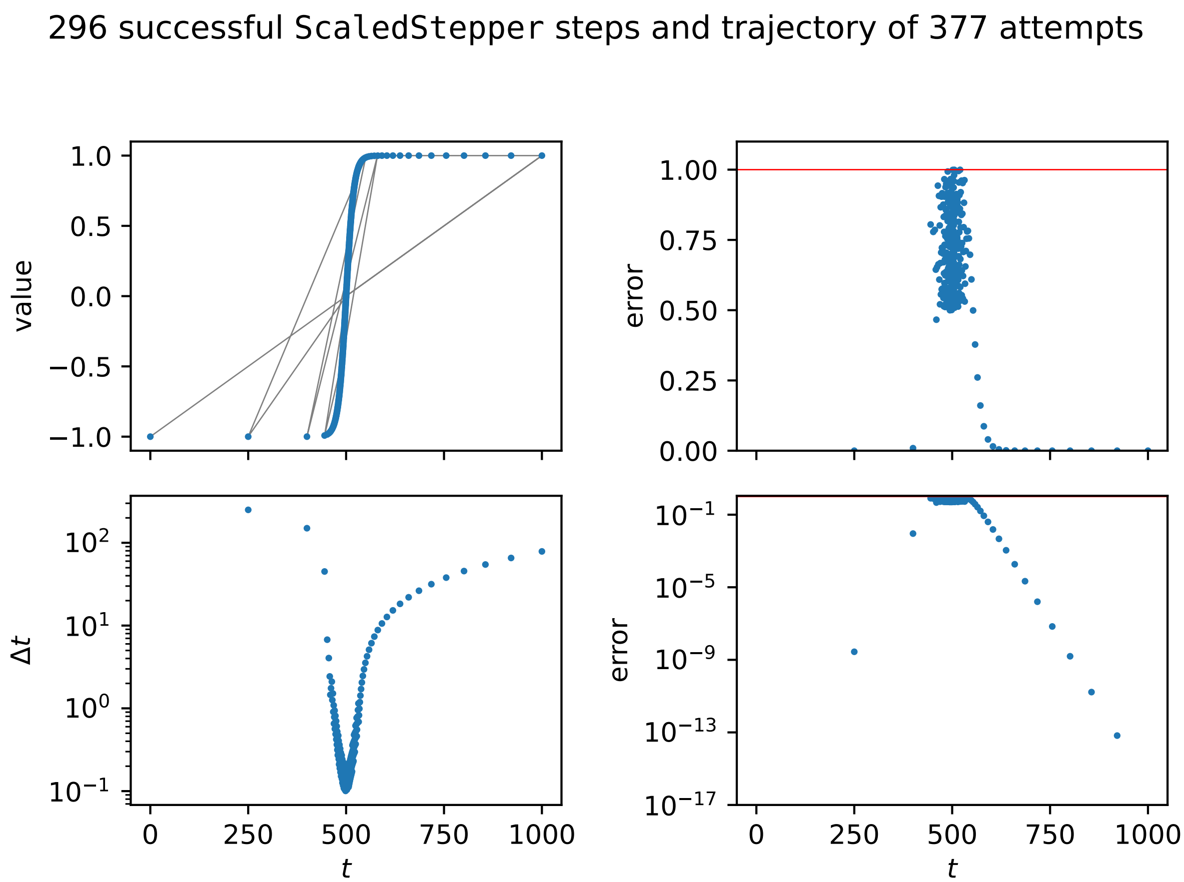
```
>>> old = -1.
>>> stepper = ScaledStepper(start=0., stop=totaltime, inclusive=True,
...                         record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     error = abs(new - old) / errorscale
...
...     if step.succeeded(value=new, error=error):
...         old = new
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
296 succesful steps in 377 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

Check that the post hoc error satisfies the desired tolerance.

```
>>> print(max(errors) < 1.)
True
```



property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to *Stepper* via *succeeded()*.

next()

Return the next step.

Note: Legacy Python 2.7 support.

Return type

Step

Raises

StopIteration – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

ndarray of values of the control variable attempted so far.

succeeded (*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns

Whether step was successful.

Return type

bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

steppyngstounes.sequenceStepper

Classes

class steppyngstounes.sequenceStepper.**SequenceStepper** (*start*, *stop*, *sizes*, *inclusive=False*, *record=False*)

Bases: *Stepper*

Stepper that takes a series of fixed steps.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **sizes** (*iterable of float*) – Desired step sizes. In the event that *start* plus the sum of *sizes* will exceed *stop*, the stepper will terminate at *stop*.
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False)

- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).

Examples

```
>>> import numpy as np
>>> from steppyngstounes import SequenceStepper
```

We'll demonstrate using an artificial function that changes abruptly, but smoothly, with time,

$$\tanh \frac{\frac{t}{t_{\max}} - \frac{1}{2}}{2w}$$

where t is the elapsed time, $t_{\max}$ is total time desired, and w is a measure of the step width.

```
>>> totaltime = 1000.
>>> width = 0.01
```

The scaled "error" will be a measure of how much the solution has changed since the last step, | *new* - *old* | / *errorscale*).

```
>>> errorscale = 1e-2
```

Iterate over the stepper from *start* to *stop* (inclusive of calculating a value at *start*).

```
>>> stepper = SequenceStepper(start=0., stop=totaltime, inclusive=True,
...                           sizes=range(1,10000), record=True)
>>> for step in stepper:
...     new = np.tanh((step.end / totaltime - 0.5) / (2 * width))
...
...     _ = step.succeeded(value=new)
```

```
>>> s = "{} succesful steps in {} attempts"
>>> print(s.format(stepper.successes.sum(),
...                len(stepper.steps)))
46 succesful steps in 46 attempts
```

```
>>> steps = stepper.steps[stepper.successes]
>>> ix = steps.argsort()
>>> values = stepper.values[stepper.successes][ix]
>>> errors = abs(values[1:] - values[:-1]) / errorscale
```

As this stepper doesn't use the error, we don't expect the post hoc error to satisfy the tolerance.

property errors

ndarray of the "error" at each step attempt.

The user-determined "error" scalar value (positive and normalized to 1) at each step attempt is passed to *Stepper* via *succeeded()*.

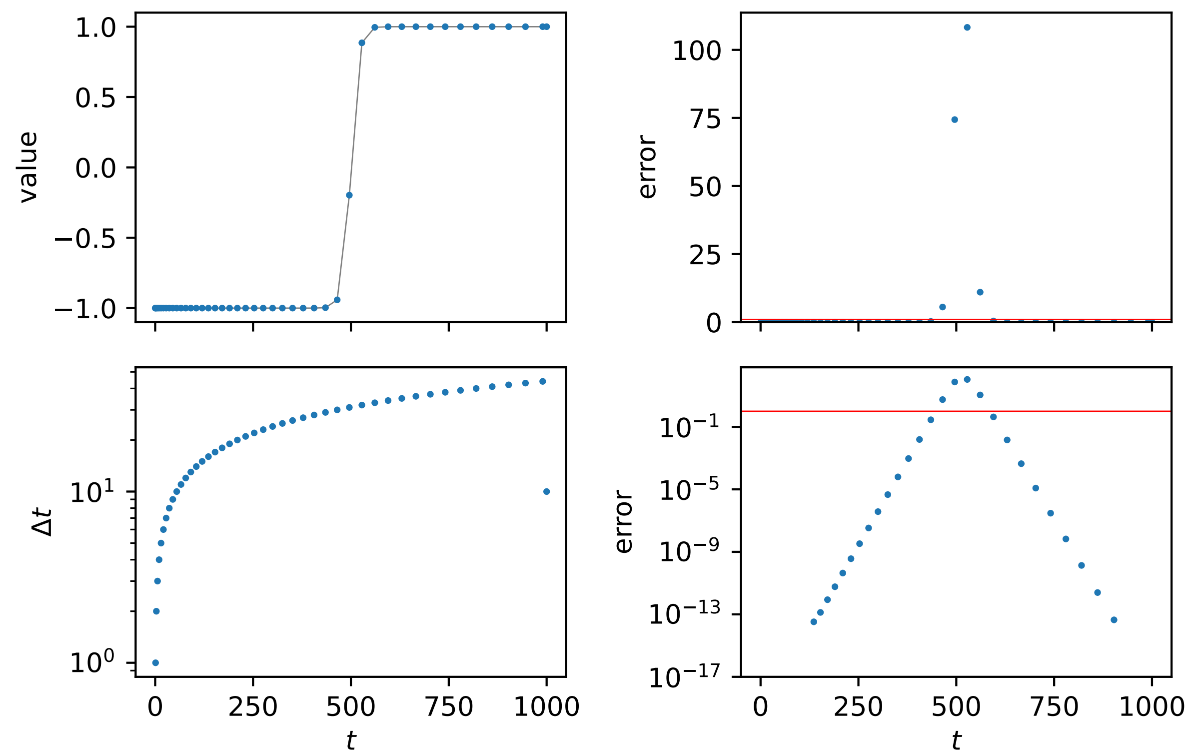
next()

Return the next step.

Note: Legacy Python 2.7 support.

---

46 successful SequenceStepper steps and trajectory of 46 attempts



Return type
Step

Raises
StopIteration – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

ndarray of values of the control variable attempted so far.

succeeded (*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns
Whether step was successful.

Return type
bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

steppyngstounes.stepper

Classes

class steppyngstounes.stepper.**Step** (*begin*, *end*, *stepper*, *want*)

Bases: object

Object describing a step to take.

Parameters

- **begin** (*float*) – The present value of the variable to step over.
- **end** (*float*) – The desired value of the variable to step over.
- **stepper** (*Stepper*) – The adaptive stepper that generated this step.
- **want** (*float*) – The step size really desired if not constrained by, e.g., end of range.

succeeded (*value=None*, *error=None*)

Test if step was successful.

Parameters

- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which `Stepper` is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which `Stepper` is being used. (default None).

Returns

Whether step was successful. If *error* is not required, returns *True*.

Return type

bool

---

class steppyngstounes.stepper.**Stepper** (*start*, *stop*, *size=None*, *minStep=None*, *inclusive=False*, *record=False*, *limiting=False*)

Bases: `object`

Adaptive stepper base class.

Parameters

- **start** (*float*) – Beginning of range to step over.
- **stop** (*float*) – Finish of range to step over.
- **size** (*float*) – Suggested step size to try (default None).
- **minStep** (*float*) – Smallest step to allow (default *(stop - start)* * *eps* (https://numpy.org/doc/stable/reference/generated/numpy.finfo.html)).
- **inclusive** (*bool*) – Whether to include an evaluation at *start* (default False).
- **record** (*bool*) – Whether to keep history of steps, errors, values, etc. (default False).
- **limiting** (*bool*) – Whether to prevent error from exceeding 1 (default False).

property errors

ndarray of the “error” at each step attempt.

The user-determined “error” scalar value (positive and normalized to 1) at each step attempt is passed to `Stepper` via `succeeded()`.

next ()

Return the next step.

Note: Legacy Python 2.7 support.

Return type

`Step`

Raises

StopIteration – If there are no further steps to take

property sizes

ndarray of the step size at each step attempt.

property steps

ndarray of values of the control variable attempted so far.

succeeded (*step*, *value=None*, *error=None*)

Test if step was successful.

Stores data about the last step.

Parameters

- **step** (*Step*) – The step to test.
- **value** (*float, optional*) – User-determined scalar value that characterizes the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).
- **error** (*float, optional*) – User-determined error (positive and normalized to 1) from the last step. Whether this parameter is required depends on which *Stepper* is being used. (default None).

Returns
Whether step was successful.

Return type
bool

property successes

ndarray of whether the step was successful at each step attempt.

property values

ndarray of the "value" at each step attempt.

The user-determined scalar value at each step attempt is passed to *Stepper* via *succeeded()*.

7 Terms of Use

BY PERSONS OR PROPERTY OR OTHERWISE, AND WHETHER OR NOT LOSS WAS SUSTAINED FROM, OR AROSE OUT OF THE RESULTS OF, OR USE OF, THE SOFTWARE OR SERVICES PROVIDED HEREUNDER.

Python Module Index

S

Index

V