

Using a Tolerant Cartesian Vector Class

Copyright (c) 2019 Tor Olav Kristensen, <http://subcube.com> (<http://subcube.com>)

<https://github.com/t-o-k/scikit-vectors> (<https://github.com/t-o-k/scikit-vectors>)

Use of this source code is governed by a BSD-license that can be found in the LICENSE file.

```
In [1]: 1 from skvectors import create_class_Tolerant_Cartesian_Vector
```

```
In [2]: 1 # Create a 4-dimensional tolerant cartesian vector class
2
3 TCVC = create_class_Tolerant_Cartesian_Vector('TCVC', 'ABCD')
4
5 # Explicit alternative:
6 # TCVC = \
7 #     create_class_Tolerant_Cartesian_Vector(
8 #         name = 'TCVC',
9 #         component_names = [ 'A', 'B', 'C', 'D' ],
10 #         brackets = [ '<', '>' ],
11 #         sep = ', ',
12 #         cnull = 0,
13 #         cunit = 1,
14 #         functions = None,
15 #         abs_tol = 1e-12,
16 #         rel_tol = 1e-9
17 #     )
```

```
In [3]: 1 # Absolute tolerance for vector lengths
2       2 TCVC.abs_tol
```

Out[3]: 1e-12

```
In [4]: 1 # Relative tolerance for vector lengths
2       2 TCVC.rel_tol
```

Out[4]: 1e-09

```
In [5]: 1 # Calculate the tolerance for a vector based on its length
        2 u = TCVC(0.0, 0.0, 0.0, 0.0) # u.length() = 0.0
        3 u.tolerance(), u.tol
```

Out[5]: (1e-12, 1e-12)

```
In [6]: 1 # Calculate the tolerance for a vector based on its length
        2 u = TCVC(-0.2, 0.4, 0.8, -0.4) # u.length() = 1.0
        3 u.tol, (1e6 * u).tol
```

Out[6]: (1e-09, 0.001)

```
In [7]: 1 # Calculate the tolerance for a vector based on its length
        2 u = TCVC(-1, 2, 4, -2) # u.length() = 5.0
        3 u.tol, (u / 1e3).tol, (u / 1e6).tol, (u / 1e9).tol
```

Out[7]: (5e-09, 5.0000000000000005e-12, 1e-12, 1e-12)

```
In [8]: 1 # Calculate a common tolerance for a vector and another based on their lengths
        2 u = TCVC(0, 0, 0, 0)
        3 v = TCVC(0, 0, 0, 0)
        4 u.tolerance_with(v)
```

Out[8]: 1e-12

```
In [9]: 1 # Calculate a common tolerance for a vector and another based on their lengths
        2 u = TCVC(-0.2, 0.4, 0.8, -0.4) # u.length() = 1.0
        3 v = TCVC(-1.0, 2.0, 4.0, -2.0) # v.length() = 5.0
        4 u.tolerance_with(v), v.tolerance_with(u)
```

Out[9]: (5e-09, 5e-09)

```
In [10]: 1 # Calculate a common tolerance for several vectors based on their lengths
         2 u = TCVC(0, 0, 0, 0)
         3 v = TCVC(0, 0, 0, 0)
         4 some_vectors = [ u, v ]
         5 TCVC.tolerance_all(some_vectors)
```

Out[10]: 1e-12

```
In [11]: 1 # Calculate a common tolerance for several vectors based on their lengths
2 u = TCVC(-0.2, 0.4, 0.8, -0.4) # u.length() = 1.0
3 v = TCVC(-1.0, 2.0, 4.0, -2.0) # v.length() = 5.0
4 some_vectors = [ u, v, u - v, u + v ]
5 TCVC.tolerance_all(some_vectors), TCVC.tolerance_all(vector for vector in some_vectors)
```

Out[11]: (6.0000000000000001e-09, 6.0000000000000001e-09)

```
In [12]: 1 # NB: This does not work:
2 # TCVC.tolerance_all([ ])
```

```
In [13]: 1 # NB: This does not work:
2 # u = TCVC(-1.0, 2.0, 4.0, -2.0)
3 # TCVC.tolerance_all([ u ])
```

```
In [14]: 1 # Check if the length of a vector is equal to cnull (within a calculated tolerance)
2 nil = TCVC.abs_tol / 2
3 u = TCVC(0, nil, 0, 0) # u.length() = 5e-13
4 u.is_zero_vector()
```

Out[14]: True

```
In [15]: 1 # Check if the length of a vector is equal to cnull (within a calculated tolerance)
2 not_nil = TCVC.abs_tol * 2
3 u = TCVC(0, not_nil, 0, 0) # u.length() = 2e-12
4 u.is_zero_vector()
```

Out[15]: False

```
In [16]: 1 # Check if the length of a vector is not equal to cnull (within a calculated tolerance)
2 nil = TCVC.abs_tol / 2
3 u = TCVC(0, nil, 0, 0) # u.length() = 5e-13
4 bool(u)
```

Out[16]: False

```
In [17]: 1 # Check if the length of a vector is not equal to cnull (within a calculated tolerance)
2 not_nil = TCVC.abs_tol * 2
3 u = TCVC(0, not_nil, 0, 0) # u.length() = 2e-12
4 bool(v)
```

Out[17]: True

```
In [18]: 1 # Check if the length of a vector is equal to cunit (within a calculated tolerance)
2 u = TCVC(-0.2, 0.4, 0.8, -0.4) # u.length() = 1.0
3 nil = TCVC.rel_tol / 2
4 v = (1 + nil) * u # Make the length of v slightly longer than 1.0; v.length() = 1.0 + 5e-10
5 v.is_unit_vector()
```

Out[18]: True

```
In [19]: 1 # Check if the length of a vector is equal to cunit (within a calculated tolerance)
2 u = TCVC(-0.2, 0.4, 0.8, -0.4) # u.length() = 1.0
3 not_nil = TCVC.rel_tol * 2
4 v = (1 + not_nil) * u # Make the length of v longer than 1.0; v.length() = 1.0 + 2e-9
5 v.is_unit_vector()
```

Out[19]: False

```
In [20]: 1 # Check if a vector is equal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 nil = u.tolerance() / 2
4 v = (1 + nil / u.length()) * u # Make v slightly different from u
5 u == v
```

Out[20]: True

```
In [21]: 1 # Check if a vector is equal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 not_nil = u.tolerance() * 2
4 v = (1 + not_nil / u.length()) * u # Make v different from u
5 u == v
```

Out[21]: False

```
In [22]: 1 # Check if a vector is equal to any of some other vectors (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 nil = u.tolerance() / 2
4 v = (1 + nil / u.length()) * u # Make v slightly different from u
5 w = TCVC(5, 4, 0, -3)
6 some_vectors = [ v, w ]
7 u in some_vectors
```

Out[22]: True

```
In [23]: 1 # Check if a vector is equal to other any of some other vectors (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 not_nil = u.tolerance() * 2
4 v = (1 + not_nil / u.length()) * u # Make v different from u
5 w = TCVC(5, 4, 0, -3)
6 some_vectors = [ v, w ]
7 u in some_vectors
```

Out[23]: False

```
In [24]: 1 # Check if a vector is not equal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 nil = u.tolerance() / 2
4 v = (1 + nil / u.length()) * u # Make v slightly different from u
5 u != v
```

Out[24]: False

```
In [25]: 1 # Check if a vector is not equal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 not_nil = u.tolerance() * 2
4 v = (1 + not_nil / u.length()) * u # Make v different from u
5 u != v
```

Out[25]: True

```
In [26]: 1 # Check if a vector is not equal to any of some other vectors (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 nil = u.tolerance() / 2
4 v = (1 + nil / u.length()) * u # Make v slightly different from u
5 w = TCVC(5, 4, 0, -3)
6 some_vectors = [ v, w ]
7 u not in some_vectors
```

Out[26]: False

```
In [27]: 1 # Check if a vector is not equal to any of some other vectors (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 not_nil = u.tolerance() * 2
4 v = (1 + not_nil / u.length()) * u # Make v different from u
5 w = TCVC(5, 4, 0, -3)
6 some_vectors = [ v, w ]
7 u not in some_vectors
```

Out[27]: True

```
In [28]: 1 # Check if a vector has equal length to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 nil = u.tolerance_with(v) / 2
5 v *= (1 + nil / u.length()) # Make v slightly longer
6 u.equal_lengths(v)
```

Out[28]: True

```
In [29]: 1 # Check if a vector has equal length to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 not_nil = u.tolerance_with(v) * 2
5 v *= (1 + not_nil / u.length()) # Make v longer
6 u.equal_lengths(v)
```

Out[29]: False

```
In [30]: 1 # Check if a vector is shorter than another vector (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 nil = u.tolerance_with(v) / 2
5 u *= (1 - nil / u.length()) # Make u slightly shorter
6 u.shorter(v)
```

Out[30]: False

```
In [31]: 1 # Check if a vector is shorter than another vector (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 not_nil = u.tolerance_with(v) * 2
5 u *= (1 - not_nil / u.length()) # Make u shorter
6 u.shorter(v)
```

Out[31]: True

```
In [32]: 1 # Check if a vector is longer than another vector (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 nil = u.tolerance_with(v) / 2
5 u *= (1 + nil / u.length()) # Make u slightly longer
6 u.longer(v)
```

Out[32]: False

```
In [33]: 1 # Check if a vector is longer than another vector (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(-5, -4, 3, 0)
4 not_nil = u.tolerance_with(v) * 2
5 u *= (1 + not_nil / u.length()) # Make u longer
6 u.longer(v)
```

Out[33]: True

```
In [34]: 1 # Check if a vector is orthogonal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(0, 0, 0, 0)
4 nil = TCVC.abs_tol / 2
5 v.A = nil
6 u.are_orthogonal(v)
```

Out[34]: True

```
In [35]: 1 # Check if a vector is orthogonal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(0, 0, 0, 0)
4 not_nil = TCVC.abs_tol * 2
5 v.A = not_nil
6 u.are_orthogonal(v)
```

Out[35]: False

```
In [36]: 1 # Check if a vector is orthogonal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(0, 5, 1, 3)
4 nil = 1e-12
5 v.A = nil
6 u.are_orthogonal(v), (u * 1e9).are_orthogonal(v / 1e9), (u / 1e9).are_orthogonal(v * 1e9)
```

Out[36]: (True, True, True)

```
In [37]: 1 # Check if a vector is orthogonal to another (within a calculated tolerance)
2 u = TCVC(4, -3, 0, 5)
3 v = TCVC(0, 5, 1, 3)
4 not_nil = 1e-10
5 v.A = not_nil
6 u.are_orthogonal(v), (u * 1e9).are_orthogonal(v / 1e9), (u / 1e9).are_orthogonal(v * 1e9)
```

Out[37]: (False, False, False)

```
In [38]: 1 # Create a vector by rounding the component values in a vector
2 u = TCVC(-1.0000000004, 1.0, 3.9999999996, 2.123456789) # u.tolerance() = circa 4.7e-9
3 u.round_components(), u.cround
```

Out[38]: (TCVC(A=-1.0, B=1.0, C=4.0, D=2.12345679),
TCVC(A=-1.0, B=1.0, C=4.0, D=2.12345679))

In []: 1