QuaternionAnalysis Package

User's Guide

Fernando Miranda
Maria Irene Falcão

The *Mathematica* QuaternionAnalysis package adds functionalities to the Quaternions package for implementing Hamilton's quaternion algebra. It also provides tools for manipulating regular quaternion valued functions, in the sense of Fueter. Some of the added new features include the possibility of performing operations on functions defined in $\mathbb{R}^{n+1}$, $n \geq 2$.

About QuaternionAnalysis Package

QuaternionAnalysis is an add-on application for manipulating quaternion valued, or more generally paravector valued, regular functions. It is closely based on the book *Holomorphic functions in the plane and n-dimensional space*, by K. Guerlebeck, K. Habetha and W. Sproessig.

Version: 1.0.0

Date: November 2014

Download: The first version of the package, informations about installation and future updates can be obtained from the website

`http://w3.math.uminho.pt/QuaternionAnalysis`

Requirements: *Mathematica* Quaternions Package

Warnings:

- The QuaternionAnalysis package adds functionality to the following functions: `Plus`, `Times`, `Divide`, `Power`, `Re`, `Conjugate`, `Dot`, `Abs`, `Norm`, `Sign` and `Derivative`.
- In this package, the quaternions can have real valued or symbolic entries.
- Some of the new functions overshadow those of the Quaternions package: `Abs`, `Norm`, `ScalarQ`, ``Quaternions`Private`extfunc``.

Keywords: Fueter, Hamilton, quaternions, paravectors, monogenic functions.

Tutorial

This package endows the standard package Quaternions which implements Hamilton's quaternion algebra with the ability to perform operations on quaternion-valued functions.

A collection of new functions is introduced to provide basic mathematical tools necessary for dealing with quaternion valued regular functions, in the sense of Fueter. Some of the new added features include the possibility of manipulating functions defined in $\mathbb{R}^{n+1}$, for $n \geq 2$.

Hamilton's Quaternion Algebra

A quaternion x is a number of the form

$$x = x_0 + i\,x_1 + j\,x_2 + k\,x_3,\ \ x_i \in \mathbb{R},$$

where i, j and k satisfy the multiplication rules:

$$i^2 = j^2 = k^2 = i\,j\,k = -1.$$

This noncommutative product generates the algebra of real quaternions $\mathbb{H}$. The real vector space $\mathbb{R}^4$ can be embedded in $\mathbb{H}$ by identifying the element $x = (x_0, x_1, x_2, x_3) \in \mathbb{R}^4$ with the element $x = x_0 + i\,x_1 + j\,x_2 + k\,x_3 \in \mathbb{H}$.

The standard package Quaternions adds rules to `Plus`, `Minus`, `Times`, `Divide` and the fundamental `NonCommutativeMultiply`. Among others, the following quaternion functions are included: `Re`, `Conjugate`, `AbsIJK`, `Sign`, `AdjustedSignIJK`, `ToQuaternion`, `FromQuaternion` and `QuaternionQ`.

In the QuaternionAnalysis package, a quaternion is also an object of the form `Quaternion[x0,x1,x2,x3]`, whose entries are not necessarily numeric valued. This package allows the use of symbolic entries, assuming that all symbols represent real numbers.

Quaternion[x0, x1, x2, x3]	the quaternion x0+i x1+j x2+k x3 with symbolic entries
Vec[x]	the vector part of x
Abs[x]	extends the absolut value function to quaternion objects
AbsVec[x]	the absolute value of the vector part of x
Norm[x]	extends the norm function to quaternion objects
W[x]	the sign of the vector part of the quaternion x
Dot[x, y]	extends the dot product to quaternion objects
SymmetricProduct[x, y]	the symmetric product of two quaternions
QPower[x, n] x^n	a recursive implementation of the Power
PureQuaternionQ[x]	gives True if x is a pure quaternion
ScalarQ[x]	overloads the original ScalarQ to allow symbolic entries

Quaternion Algebra

This loads the package

```
<< QuaternionAnalysis`
```

⚠ SetCoordinates::valid : The coordinates system is set to {X0, X1, X2, X3}.

Quaternions can have either numeric entries or symbolic ones

```
Quaternion[1, 2, 3, 4] + 2 Quaternion[a, b, c, d] // TraditionalForm
```

$1 + 2 a + (2 + 2 b)\, ⅈ + (3 + 2 c)\, ⅉ + (4 + 2 d)\, 𝕜$

Operations on Quaternions

```
x = Quaternion[1, 2, 3, 4]; y = Quaternion[4, 3, 2, 1];
x ** Vec[x]
```

```
Quaternion[-29, 2, 3, 4]
```

```
SymmetricProduct[x, y]
```

```
Quaternion[-12, 11, 14, 17]
```

```
(x ** y + y ** x) / 2
```

```
Quaternion[-12, 11, 14, 17]
```

```
QPower[Quaternion[a, b, c, d], 2]
```

$\text{Quaternion}\left[a^2 - b^2 - c^2 - d^2,\ 2\, a\, b,\ 2\, a\, c,\ 2\, a\, d\right]$

`W` is the sign of the vector part of a quaternion. If x is a real number, $\omega(x) = 0$

```
W[x] // TraditionalForm
```

$$\frac{2\, ⅈ}{\sqrt{29}} + \frac{3\, ⅉ}{\sqrt{29}} + \frac{4\, 𝕜}{\sqrt{29}}$$

```
Sign[Vec[x]]
```

$$\text{Quaternion}\left[0,\ \frac{2}{\sqrt{29}},\ \frac{3}{\sqrt{29}},\ \frac{4}{\sqrt{29}}\right]$$

```
W[Quaternion[a, 0, 0, 0]]
```

```
Quaternion[0, 0, 0, 0]
```

◻ Representation of real quaternions

■ *Complex-like form*

A non-real quaternion x can be written as

$$x = x_0 + \omega(x)\, |\underline{x}|,$$

where x_0 and $\underline{x}$ are the real and vector parts of x, respectively, and $\omega(x)$ is the unit quaternion

$$\omega(x) = \frac{\underline{x}}{|\underline{x}|},$$

very much like a complex number is written in the form $a + i\, b$. This representation is known as the complex-like form of a quaternion.

To perform operations on quaternions written in the complex-like form, an object of the form `ComplexLike[a,b]` is defined. For such objects, simple rules as `Plus, Times, Power` and functions as `Re, Abs, Norm`, etc. are defined, mimicking the complex ones.

- ***Polar representation***

A non-real quaternion x permits the representation

$$x = |x|(\cos\phi + \omega(x)\sin\phi),$$

where ϕ =arccot $\frac{x_0}{|\underline{x}|}$

- ***Complex representation***

Any quaternion $x = x_0 + i\, x_1 + j\, x_2 + k\, x_3$ can be written as

$$x = (x_0 + i\, x_1) + (x_2 + i\, x_3)\, j$$

and represented by the complex matrix

$$\begin{pmatrix} x_0 + i\, x_1 & x_2 + i\, x_3 \\ -x_2 + i\, x_3 & x_0 - i\, x_1 \end{pmatrix}.$$

Thus, the product of two quaternions can be expressed by the product of the two corresponding complex matrices.

- ***Matrix representation***

The left and right representations of the quaternion $x = x_0 + i\, x_1 + j\, x_2 + k\, x_3$ are

$$L_x = \begin{pmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & -x_3 & x_2 \\ x_2 & x_3 & x_0 & -x_1 \\ x_3 & -x_2 & x_1 & x_0 \end{pmatrix}$$

and

$$R_x = \begin{pmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & x_3 & -x_2 \\ x_2 & -x_3 & x_0 & x_1 \\ x_3 & x_2 & -x_1 & x_0 \end{pmatrix}.$$

These matrices can be associated with the left and right product of the quaternions x and y, i.e

$$x \to L_x \text{ with } x\, y \to L_x\, \mathbf{y} \text{ and } x \to R_x \text{ with } y\, x \to R_x\, \mathbf{y}$$

where $\mathbf{y}$ is the vector in $\mathbb{R}^4$ corresponding to the quaternion y.

ComplexLike[x,y]	the real part and the norm of the vector part of a quaternion
ToComplexLike[x]	complex-like representation of a quaternion
PolarForm[x]	polar form of a quaternion

QuaterniontoComplex[x]	complex representation of a quaternion
ComplexToQuaternion[x,y]	returns the quaternion $x + y J$
QuaternionToComplexMatrix[x]	complex representation matrix of a quaternion
QuaternionToMatrixR[x]	real right representation matrix of a quaternion
QuaternionToMatrixL[x]	real left representation matrix of a quaternion

Representation of real quaternions

`ToComplexLike` gives the complex-like form of a quaternion

```
x = Quaternion[1, 2, 3, 4];
X = ToComplexLike[x]
```

```
ComplexLike[1, √29]
```

```
X // TraditionalForm
```

$1 + \sqrt{29}\, w$

Operations on `ComplexLike` objects

```
Conjugate[X]
```

```
ComplexLike[1, -√29]
```

```
Abs[X]
```

$\sqrt{30}$

If $\omega(x) = \omega(y)$, then all the algebraic operations can be computed as if x and y were complex numbers.

```
y = Quaternion[a, 2, 3, 4];
x ** y
```

```
Quaternion[-29 + a, 2 + 2 a, 3 + 3 a, 4 + 4 a]
```

```
ToComplexLike[x] ToComplexLike[y]
```

```
ComplexLike[-29 + a, √29 + √29 a]
```

```
First[%] + Last[%] W[x] // Simplify
```

```
Quaternion[-29 + a, 2 + 2 a, 3 + 3 a, 4 + 4 a]
```

The PolarForm of a quaternion

```
PolarForm[x]
```

$\left\{\sqrt{30}, \text{ArcCot}\left[\frac{1}{\sqrt{29}}\right]\right\}$

```
{Abs[x], ArcCot[Re[x] / AbsVec[x]]}
```

$\left\{\sqrt{30}, \text{ArcCot}\left[\frac{1}{\sqrt{29}}\right]\right\}$

Complex form of a quaternion

```
QuaternionToComplex[x]
```

```
{1 + 2 i, 3 + 4 i}
```

```
ComplexToQuaternion @@ %
```

```
Quaternion[1, 2, 3, 4]
```

```
QuaternionToComplexMatrix[x] // MatrixForm
```

$$\begin{pmatrix} 1+2i & 3+4i \\ -3+4i & 1-4i \end{pmatrix}$$

```
y = Quaternion[4, 3, 2, 1]; x ** y
```

```
Quaternion[-12, 6, 24, 12]
```

```
QuaternionToComplexMatrix[x].QuaternionToComplexMatrix[y] //
 MatrixForm
```

$$\begin{pmatrix} -12+6i & 24+12i \\ -24+12i & -12-6i \end{pmatrix}$$

Matrix form of a quaternion

```
QuaternionToMatrixL[x] // MatrixForm
```

$$\begin{pmatrix} 1 & -2 & -3 & -4 \\ 2 & 1 & -4 & 3 \\ 3 & 4 & 1 & -2 \\ 4 & -3 & 2 & 1 \end{pmatrix}$$

```
QuaternionToMatrixL[x].List @@ y
```

```
{-12, 6, 24, 12}
```

```
QuaternionToMatrixR[y].List@@x
```

```
{-12, 6, 24, 12}
```

Quaternion Analysis

In 1935, R. Fueter, one of the founders of Quaternion (or Quaternionic) Analysis, proposed a generalization of complex analyticity to the quaternionic case by means of an analogue of the Cauchy-Riemann equations.

The QuaternionAnalysis package leads with quaternion-valued functions f of one quaternion variable x, defined in a domain $\Omega \subset \mathbb{R}^4$, of the form

$$f(x) = f_0(x) + i\,f_1(x) + j\,f_2(x) + k\,f_3(x),$$

where x=(x_0, x_1, x_2, x_3) and f_k are real valued in Ω functions. Continuity, differentiability or integrability are defined coordinatewisely.

The quaternionic Cauchy-Riemann operator is defined as

$$\overline{\partial} = \partial_0 + \partial_{\underline{x}}, \text{ where } \partial_0 = \frac{\partial}{\partial_{x_0}} \text{ and } \partial_{\underline{x}} \text{ is the Dirac operator } \partial_{\underline{x}} = i\frac{\partial}{\partial_{x_1}} + j\frac{\partial}{\partial_{x_2}} + k\frac{\partial}{\partial_{x_3}}.$$

A C^1-function f satisfying the equation $\overline{\partial} f = 0$ (resp. $f\,\overline{\partial} = 0$) is called left monogenic (resp. right monogenic). A function which is both left and right monogenic is called monogenic. In such case, the derivative f' can be expressed by the real partial derivatives as in the complex case:

$$f' = \frac{1}{2}\,\partial f = \partial_0 f, \text{ where } \partial = \partial_0 - \partial_{\underline{x}}$$

is the conjugate Cauchy-Riemann operator.

CauchyRiemannL[f]	Cauchy Riemann operator acting on f from the left
CauchyRiemannR[f]	Cauchy Riemann operator acting on f from the right
DiracL[f]	Dirac operator acting on f from the left
DiracR[f]	Dirac operator acting on f from the right
Laplace[f]	Laplace operator
MonogenicQ[f]	gives True for monogenic functions f
Derivative[f]	gives the derivative of a monogenic function f

Quaternion Analysis

A non-monogenic function

```
f = QPower[Quaternion[X0, X1, X2, X3], 2]
```

$$\text{Quaternion}\left[X0^2 - X1^2 - X2^2 - X3^2,\ 2\,X0\,X1,\ 2\,X0\,X2,\ 2\,X0\,X3\right]$$

```
CauchyRiemannL[f]
```

```
Quaternion[-4 X0, 0, 0, 0]
```

```
MonogenicQ[f]

False
```

Monogenic functions

```
f = Quaternion[X1 X2 X3 - 1, -X0 X2 X3 + 1, -X0 X1  X3 + 1, -X0 X1 X2 + 1]

Quaternion[-1 + X1 X2 X3, 1 - X0 X2 X3, 1 - X0 X1 X3, 1 - X0 X1 X2]

MonogenicQ[f]

True

Derivative[f]

Quaternion[0, -X2 X3, -X1 X3, -X1 X2]
```

$$g = \text{ComplexLike}\left[-\frac{r^2}{3} + x0^2,\ \frac{2\,r\,x0}{3}\right];$$

```
MonogenicQ[g, x0, r]

True
```

Clifford Analysis

Let $\{e_1, e_2, \ldots, e_n\}$ be an orthonormal basis of the euclidean vector space $\mathbb{R}^n$ with a product according to the multiplication rules

$$e_k\, e_l + e_l\, e_k = -2\,\delta_{kl},\ k,\ l = 1, \ldots, n, \text{ where } \delta_{kl} \text{ is the Kronecker symbol.}$$

This non-commutative product generates the 2^n-dimensional Clifford Algebra $Cl_{0,n}$ over $\mathbb{R}$ and the set $\{e_A : A \subset \{1, \ldots, n\}\}$ with $e_A = e_{h_1} e_{h_2} \ldots e_{h_r}$, $1 \le h_1 < \ldots < h_r \le n$, $e_\emptyset = e_0 = 1$, forms a basis of $Cl_{0,n}$. Denoting by A_n the subset of the Algebra $Cl_{0,n}$, $A_n = \text{span}_{\mathbb{R}}\{1, e_1, e_2, \ldots, e_n\}$ the real vector space $\mathbb{R}^{n+1}$ can be embedded in A_n by the identification of each element $(x_0, x_1, \ldots, x_n) \in \mathbb{R}$ with the so-called paravector $x = x_0 + x_1\, e_1 + x_2\, e_2 + \ldots + x_n\, e_n \in A_n$.
Similarly to the quaternionic and complex case, a paravector can be written in terms of a real part and a vector part as

$$x = x_0 + \underline{x},$$

the conjugate of x is

$$\overline{x} = x_0 - \underline{x}$$

and the norm $|x|$ of x is defined by

$$|x|^2 = x\,\overline{x} = x_0^2 + x_1^2 + .. + x_n^2$$

Moreover, denoting by

$$\omega(x) = \frac{x}{|\underline{x}|} \in S^n,$$

where S^n is the unit sphere in $\mathbb{R}^n$, each paravector x can be written as a complex-like number

$$x = x_0 + \omega(x)\,|\underline{x}|.$$

In general, due to the algebraic properties of $Cl_{0,n}$, we have to assume that a monogenic function f, defined in some open subset $\Omega \subset \mathbb{R}^{n+1}$, has values in $Cl_{0,n}$, i.e., it is of the form

$$f(x) = \sum_A e_A f_A(x),$$

where f_A are real functions.

The Cauchy-Riemann operator in $\mathbb{R}^{n+1}$ is obtained from the generalized Dirac operator

$$\overline{\partial} = \partial_0 + \partial_{\underline{x}}, \text{ where } \partial_{\underline{x}} = \sum_{i=1}^{n} e_i \frac{\partial}{\partial_{x_i}}$$

and f is a monogenic function in the sense of Clifford Analysis if it belongs to the kernel of $\overline{\partial}$.
In such case,

$$f' = \frac{1}{2}\,\partial f = \partial_0 f,$$

like in the complex and quaternionic case.

One of the objectives of the QuaternionAnalysis package is to endow the original Quaternions package with the ability to operate on paravector elements. For this purpose, the new object `Paravector` is introduced and some elementary operations and functions are extended: `ToComplexLike`, `Plus`, `Re`, `Conjugate`, `Dot`, `Abs`, `Norm`, `Vec`, `AbsVec`, `W`, `MonogenicQ`, `Derivative`.

If nothing is declare otherwise, it is assumed that $n = 3$ and therefore the new functions accept as arguments objects of the form Quaternion or `Paravector`. For different values of n one has to declare the space dimension, through the command `SetCoordinates`.

Paravector[x0,x1,x2, ...]	represents the paravector x0 + x1 e1 + x2 e2 + ...
SetCoordinates[x0,x1,x2, ...]	sets the default coordinates to be x0, x1, x2, ...
SetCoordinates[n]	sets the default coordinates to be X0, X1, ... , X_{n-1}
$CoordinatesList	gives the list of variables. {X0, X1, X2, X3} is the default list
$Dim	gives the dimension of the space. The default dimension is 4 (n=3)

Paravectors

Set the dimension of the space

```
SetCoordinates[x0, x1, x2]
```

SetCoordinates::valid : The coordinates system is set to {x0, x1, x2}.

```
$Dim
```

3

Operations on paravectors

```
(p = Paravector[1, 1, 0]) // TraditionalForm
```

$e_0 + e_1$

```
(q = Paravector[-1, 0, 1]) // TraditionalForm
```

$-e_0 + e_2$

```
Conjugate[p] + Vec[q]
```

```
Paravector[1, -1, 1]
```

```
ToComplexLike[p]
```

```
ComplexLike[1, 1]
```

The product of two paravectors in A_n is an element of $Cl_{0,n}$ but, in general, it is not an A_n-element. Hence, the multiplication is not extended for this class of objects.

```
p ** q
```

```
Paravector[1, 1, 0] ** Paravector[-1, 0, 1]
```

```
p + 2 q
```

```
Paravector[-1, 1, 2]
```

Paravector-valued functions

```
p = Paravector[x0^2 - x1^2 - x2^2, x0 x1, x0 x2];
MonogenicQ[p]
```

```
False
```

```
Derivative[p]
```

```
MonogenicQ::Fail : The function is non monogenic.
```

```
q = Paravector[x0^2 - x1^2/2 - x2^2/2, x0 x1, x0 x2];
MonogenicQ[q]
```

```
True
```

```
Derivative[q]
```

```
Paravector[2 x0, x1, x2]
```

Special Monogenic Polynomials

The QuaternionAnalysis package includes several functions to construct special monogenic paravector-valued polynomials which behave like monomial functions in the sense of the complex powers $z^k = (x_0 + i\,x_1)^k,\ k = 1, 2, \ldots$ These polynomials are of the form

$$P_k^n(x) = \sum_{s=0}^{k} T_s^k\, x^{k-s}\, \overline{x}^s,\ x \in \mathbb{R}^{n+1}$$

where T_s^k are the numbers

$$T_s^k = \frac{k!}{(n)_k} \frac{\left(\frac{n+1}{2}\right)_{k-s} \left(\frac{n-1}{2}\right)_s}{(k-s)!\, s!}$$

($(a)_r$ is the Pochhammer symbol).

These monomial-like polynomials can be wriitten in several other forms, namely in terms of the:

real and vector part of x

$$P_k^n(x) = \sum_{s=0}^{k} \binom{k}{s} x_0{}^{k-s}\, P_k^n(\underline{x}) = \sum_{s=0}^{k} \binom{k}{s} c_s(n)\, x_0{}^{k-s}\, \underline{x}^s$$

where

$$c_s(n) = \sum_{t=0}^{s} (-1)^t\, T_t^s = \begin{cases} \frac{s!!\,(n-2)!!}{(n+s-1)!!} & \text{if } s \text{ is odd} \\ c_{s-1}(n) & \text{if } s \text{ is even} \end{cases} \text{ and } c_0(n) = 1,\ n \geq 0$$

(!! is the Factorial2)

complex-like form

$$P_k^n(x) = u(x_0, r) + \omega(x)\, v(x_0, r)$$

where $r = |\underline{x}|$,

$$u(x_0, r) = \sum_{s=0}^{\lfloor k/2 \rfloor} \binom{k}{2\,s} (-1)^s\, c_{2\,s}(n)\, x_0{}^{k-2\,s}\, r^{2\,s}$$

and

$$v(x_0, r) = \sum_{s=0}^{\lfloor (k-1)/2 \rfloor} \binom{k}{2\,s+1} (-1)^s\, c_{2\,s+1}(n)\, x_0{}^{k-2\,s-1}\, r^{2\,s+1}$$

($\lfloor \ldots \rfloor$ is the Floor function).

Tks[k,s,n]	the (k,s) element of a triangle associated with Pk
Tks[k,s]	the default case: Tks[k,s]=Tks[k,s,$Dim-1]
Ck[k,n]	the alternating sum of Tks
Ck[k]	the default case: Ck[k]=Ck[k,$Dim-1]
Pk[k,x]	special monogenic polynomial of degree k, for quaternions and 3D paravectors
Pk[k,n,x]	special monogenic polynomial of degree k, for complex-like objects

Special Polynomials

The first quaternion-valued polynomials

```
SetCoordinates[x0, x1, x2, x3]
x = Quaternion[x0, x1, x2, x3];
TableForm[
 Table[{StringJoin[{"Pk[", ToString[k], ",x]="}], Pk[k, x]},
  {k, 0, 2}]]
```

⚠ SetCoordinates::valid : The coordinates system is set to {x0, x1, x2, x3}.

Pk[0,x]= Quaternion[1, 0, 0, 0]

Pk[1,x]= Quaternion$\left[x0, \frac{x1}{3}, \frac{x2}{3}, \frac{x3}{3}\right]$

Pk[2,x]= Quaternion$\left[x0^2 - \frac{x1^2}{3} - \frac{x2^2}{3} - \frac{x3^2}{3}, \frac{2\,x0\,x1}{3}, \frac{2\,x0\,x2}{3}, \frac{2\,x0\,x3}{3}\right]$

If $n = 2$, the function Pk accepts as argument x a Paravector. The first polynomials in $\mathbb{R}^3$:

```
SetCoordinates[x0, x1, x2]; x = Paravector[x0, x1, x2];
TableForm[
 Table[{StringJoin[{"Pk[", ToString[k], ",x]="}], Pk[k, x]},
  {k, 0, 2}]]
```

⚠ SetCoordinates::valid : The coordinates system is set to {x0, x1, x2}.

Pk[0,x]= Paravector[1, 0, 0]

Pk[1,x]= Paravector$\left[x0, \frac{x1}{2}, \frac{x2}{2}\right]$

Pk[2,x]= Paravector$\left[x0^2 - \frac{x1^2}{2} - \frac{x2^2}{2}, x0\,x1, x0\,x2\right]$

The function Pk accepts also as argument x a ComplexLike object. The first polynomials in $\mathbb{R}^5$:

```
TableForm[
 Table[{StringJoin[{"Pk[", ToString[k], ",4,x]="}],
   Pk[k, 4, ComplexLike[X0, R]]}, {k, 0, 3}]]
```

Pk[0,4,x]= ComplexLike[1, 0]

Pk[1,4,x]= ComplexLike$\left[X0, \frac{R}{4}\right]$

Pk[2,4,x]= ComplexLike$\left[-\frac{R^2}{4} + X0^2, \frac{R\,X0}{2}\right]$

Pk[3,4,x]= ComplexLike$\left[-\frac{3\,R^2\,X0}{4} + X0^3, -\frac{1}{8}\,R\,\left(R^2 - 6\,X0^2\right)\right]$

Pk[k,n,x], k=0,1,... are monogenic polynomials

```
Table[MonogenicQ[Pk[k, x]], {k, 1, 4}]
```

{True, True, True, True}

```
Clear[a, r]
```

```
Assuming[a ∈ Reals && r ≥ 0,
 Table[MonogenicQ[Pk[k, 2, ComplexLike[a, r]], a, r], {k, 1, 4}]]
```

```
{True, True, True, True}
```

```
First[$CoordinatesList]
Table[MonogenicQ[Pk[k, 2, ComplexLike[x0, R]]], {k, 1, 4}]
```

```
x0
```

```
{True, True, True, True}
```

Pk[k,n,x], k=0,1,... is an Appell sequence, i.e. (Pk[k,n,x])' = k Pk[k-1,n,x]

```
Table[Derivative[Pk[k, x]], {k, 1, 3}] // TableForm
```

```
Paravector[1, 0, 0]
Paravector[2 x0, x1, x2]
```

$$\text{Paravector}\left[3\,x0^2 - \frac{3}{2}\left(x1^2 + x2^2\right),\ 3\,x0\,x1,\ 3\,x0\,x2\right]$$

```
Table[k Pk[k - 1, x] , {k, 1, 3}] // TableForm
```

```
Paravector[1, 0, 0]
Paravector[2 x0, x1, x2]
```

$$\text{Paravector}\left[3\left(x0^2 - \frac{x1^2}{2} - \frac{x2^2}{2}\right),\ 3\,x0\,x1,\ 3\,x0\,x2\right]$$

The number triangles Tk[k,s,n]

```
TableForm[Table[TableForm[Table[Tks[k, s, m], {k, 0, 4 }, {s, 0, k }]],
   {m, {2, 3, 5}}], TableDirections -> Row,
 TableHeadings → {{" Tks(2)", " Tks(3)", " Tks(5)" }}]
```

Tks(2)					Tks(3)					Tks(5)				
1					1					1				
$\frac{3}{4}$	$\frac{1}{4}$				$\frac{2}{3}$	$\frac{1}{3}$				$\frac{3}{5}$	$\frac{2}{5}$			
$\frac{5}{8}$	$\frac{1}{4}$	$\frac{1}{8}$			$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$			$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$		
$\frac{35}{64}$	$\frac{15}{64}$	$\frac{9}{64}$	$\frac{5}{64}$		$\frac{2}{5}$	$\frac{3}{10}$	$\frac{1}{5}$	$\frac{1}{10}$		$\frac{2}{7}$	$\frac{12}{35}$	$\frac{9}{35}$	$\frac{4}{35}$	
$\frac{63}{128}$	$\frac{7}{32}$	$\frac{9}{64}$	$\frac{3}{32}$	$\frac{7}{128}$	$\frac{1}{3}$	$\frac{4}{15}$	$\frac{1}{5}$	$\frac{2}{15}$	$\frac{1}{15}$	$\frac{3}{14}$	$\frac{2}{7}$	$\frac{9}{35}$	$\frac{6}{35}$	$\frac{1}{14}$

The coefficients Ck[k,2] are the generalized central binomial coefficients with weight $\frac{1}{2^k}$

```
Table[Ck[k, 2], {k, 0, 10}]
```

$$\left\{1,\ \frac{1}{2},\ \frac{1}{2},\ \frac{3}{8},\ \frac{3}{8},\ \frac{5}{16},\ \frac{5}{16},\ \frac{35}{128},\ \frac{35}{128},\ \frac{63}{256},\ \frac{63}{256}\right\}$$

```
Table[1/2^k Binomial[k, Floor[k/2]], {k, 0, 10}]
```

$$\left\{1, \frac{1}{2}, \frac{1}{2}, \frac{3}{8}, \frac{3}{8}, \frac{5}{16}, \frac{5}{16}, \frac{35}{128}, \frac{35}{128}, \frac{63}{256}, \frac{63}{256}\right\}$$

For the default dimension n=3 ($Dim=4), the coefficients Ck[k,3] = Ck[k] have the simple form

```
SetCoordinates[4]
```

SetCoordinates::valid : The coordinates system is set to {X0, X1, X2, X3}.

```
Ck[k] // Factor
```

$$\frac{3 + (-1)^k + 2 k}{2 (1 + k) (2 + k)}$$

```
Table[Ck[k], {k, 0, 10}]
```

$$\left\{1, \frac{1}{3}, \frac{1}{3}, \frac{1}{5}, \frac{1}{5}, \frac{1}{7}, \frac{1}{7}, \frac{1}{9}, \frac{1}{9}, \frac{1}{11}, \frac{1}{11}\right\}$$

Overview

Objects and other features

Quaternion — represents a quaternion by its numeric or symbolic coefficients

Paravector — represents a paravector by its coefficients

ComplexLike — represents a quaternion or a paravector by its real part and the norm of its vector part.

SetCoordinates — sets the coordinates system

$CoordinatesList $Dim

Representation forms

PolarForm — polar form of a quaternion/paravector

QuaternionToComplex — complex representation of a quaternion

QuaternionToComplexMatrix — complex representation matrix of a quaternion

QuaternionToMatrixL — real left representation matrix of a quaternion

`QuaternionToMatrixR` — real right representation matrix of a quaternion

`ToComplexLike` — representation of a quaternion/paravector by its numeric or symbolic coefficients

`ComplexToQuaternion`

□ Algebra

`Abs` — extends the absolute value function to quaternion/paravector/complexlike objects

`AbsVec` — absolute value of the vector part of an object

`Dot` — extends the dot product to quaternion objects

`Norm` — extends the norm function to quaternion/paravector/complexlike objects

`QPower` — recursive implementation of the function Power

`SymmetricProduct` — gives the symmetric product of two quaternions

`Vec` — vector part of an object

`W` — sign of the vector part of a quaternion/paravector

`PureQuaternionQ  ScalarQ`

□ Analysis

`CauchyRiemannL` — left Cauchy Riemann operator

`CauchyRiemannR` — right Cauchy Riemann operator

`DiracL` — left Dirac operator

`DiracR` — right Dirac operator

`Laplace` — Laplace operator

`LeftMonogenicQ  RightMonogenicQ  MonogenicQ`

□ Polynomials

`Pk` — special monogenic polynomials mimicking the complex powers

`Tks` — a Pascal-like triangle associated with the polinomials Pk

`Ck` — the alternating sum of Tks

References

[1] Falcão, M.I., Miranda, F.: Quaternions: A Mathematica package for quaternionic analysis. Lecture Notes in Computer Science, 6784, 200-214 (2011)

[2] Fueter, R.: Die Funktionentheorie der Differetialgleichungen Δu=0 und ΔΔu=0 mit vier reellen Variablen. Comm. Math. Helv. (7), 307-330 (1934-35)

[3] Guerlebeck, K., Habetha, K., Sproessig, W.: Holomorphic functions in the plane and n-dimensional space. Birkhauser Verlag, Basel (2008)

[4] Sudbery, A.: Quaternionic analysis. Math. Proc. Camb. Phil. Soc. 85, 199-225 (1979)

About the Authors

Fernando Miranda
CMAT & DMA, Universidade do Minho,
Campus de Gualtar
4710-057 Braga
Portugal
fmiranda@math.uminho.pt

Maria Irene Falcão
CMAT & DMA, Universidade do Minho,
Campus de Gualtar
4710-057 Braga
Portugal
mif@math.uminho.pt

Appendix - Help pages

This appendix includes the help pages and several examples for each new function included in the package QuaternionAnalysis.

AbsVec

`AbsVec[`*expr*`]`
gives the absolute value of the vector part of *expr*.

MORE INFORMATION

- To use `AbsVec`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`.
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion`, `Paravector` and `ComplexLike`.

EXAMPLES

Basic Examples (2)

```
In[1]:= Needs["QuaternionAnalysis`"]
```

Quaternion numbers:

```
In[2]:= AbsVec[Quaternion[1, 2, 3, 4]]
```

Out[2]= $\sqrt{29}$

Paravector numbers:

```
In[1]:= AbsVec[Paravector[1, 2, 3]]
```

Out[1]= $\sqrt{13}$

```
In[2]:= AbsVec[Paravector[a, b, c, d, e]]
```

Out[2]= $\sqrt{b^2 + c^2 + d^2 + e^2}$

SEE ALSO

Quaternion • Paravector • ComplexLike

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

CauchyRiemannL

`CauchyRiemannL[f]`
applies the Cauchy-Riemann operator from the left to f.

MORE INFORMATION

- To use `CauchyRiemannL`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`
- The function accepts as argument objects of the form `Quaternion`.
- CauchyRiemannL[f] = $(\partial_{x0} + I\,\partial_{x1} + J\,\partial_{x2} + K\,\partial_{x3})\,(f0 + If1 + Jf2 + Kf3)$

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= $CoordinatesList
Out[2]= {X0, X1, X2, X3}

In[3]:= CauchyRiemannL[Quaternion[X0^2 - X1^2 - X2^2 - X3^2, 2 X0 X1, 2 X0 X2, 2 X0 X3]]
Out[4]= Quaternion[-4 X0, 0, 0, 0]
```

SEE ALSO

CauchyRiemannR • DiracL • DiracR • LeftMonogenicQ • RightMonogenicQ • MonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

CauchyRiemannR

`CauchyRiemannR[f]`
applies the Cauchy-Riemann operator from the right to f.

MORE INFORMATION

- To use `CauchyRiemannR`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]
- The function accepts as argument objects of the form `Quaternion`.
- CauchyRiemannR[f] = $(f0 + I\, f1 + J\, f2 + K\, f3)\,(\partial_{x0} + I\, \partial_{x1} + J\, \partial_{x2} + K\, \partial_{x3})$

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= $CoordinatesList
Out[2]= {X0, X1, X2, X3}

In[3]:= CauchyRiemannR[Quaternion[X0^2 - X1^2 - X2^2 - X3^2, 2 X0 X1, 2 X0 X2, 2 X0 X3]]
Out[4]= Quaternion[-4 X0, 0, 0, 0]
```

SEE ALSO

CauchyRiemannL • DiracL • DiracR • LeftMonogenicQ • RightMonogenicQ • MonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Ck

Ck[k, n]
gives the alternating sum of Tk[k,s,n], i.e. Ck[k, n] $=\sum_{s=0}^{k}(-1)^{s}$ Tks[k, s, n]

Ck[k]=Ck[k, \$Dim − 1].

MORE INFORMATION

- To use Ck, you first need to load the Quaternion Analysis Package using Needs["QuaternionAnalysis`"] .
- Mathematical function suitable for both symbolic and numerical manipulation.

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
Table[Ck[k, 2], {k, 0, 10}]
```

Out[2]= $\left\{1, \frac{1}{2}, \frac{1}{2}, \frac{3}{8}, \frac{3}{8}, \frac{5}{16}, \frac{5}{16}, \frac{35}{128}, \frac{35}{128}, \frac{63}{256}, \frac{63}{256}\right\}$

In[3]:=
```
SetCoordinates[4];
Ck[k] // Factor
```

SetCoordinates::valid : The coordinates system is set to {X0, X1, X2, X3}.

Out[3]= $\frac{3+(-1)^{k}+2k}{2(1+k)(2+k)}$

SEE ALSO

Pk • Tks • Factorial2

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

ComplexLike

`ComplexLike[x, y]`
represents a quaternion or a paravector, by its real part - x- and the norm of its vector part -y

MORE INFORMATION

- To use `ComplexLike`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]

EXAMPLES

Basic Examples (2)

In[1]:= `Needs["QuaternionAnalysis`"]`

SetCoordinates::valid : The coordinates system is set to {X0, X1, X2, X3}.

`Paravector` objects:

In[2]:= `SetCoordinates[3]`

Coordinates::valid : The coordinates system is set to {X0, X1, X2}.

In[2]:= `ToComplexLike[Paravector[a, b, c]]`

Out[3]= ComplexLike$\left[a, \sqrt{b^2 + c^2}\right]$

In[4]:= `% // TraditionalForm`

Out[4]= $a + \sqrt{b^2 + c^2}\ w$

Operations on `ComplexLike` objects:

The product is defined for objects of the CompleLike form; it is similar to the complex product.

In[1]:= `(x = ComplexLike[1, 2]) // TraditionalForm`
`(y = ComplexLike[2, 3]) // TraditionalForm`

Out[1]= $1 + 2\ w$

Out[1]= $2 + 3\ w$

In[2]:= `x y // TraditionalForm`

Out[2]= $-4 + 7\ w$

In[3]:= `Conjugate[ComplexLike[1, 2]]`

Out[3]= ComplexLike[1, -2]

Possible Issues (1)

Different objects can have the same ComplexLike representation

In[1]:= `ToComplexLike[Paravector[1, 1, 5]]`

Out[1]= ComplexLike$\left[1, \sqrt{26}\right]$

In[2]:= `ToComplexLike[Quaternion[1, 3, -4, 1]]`

Out[2]= ComplexLike$\left[1, \sqrt{26}\right]$

SEE ALSO

Quaternion • Paravector • ToComplexLike • W

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

ComplexToQuaternion

ComplexToQuaternion[x,y]
gives the quaternion x+yJ.

MORE INFORMATION

- To use `ComplexToQuaternion`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]`
- Mathematical function suitable for both symbolic and numerical manipulation.
- Any quaternion $x = x_0 + i\,x_1 + j\,x_2 + k\,x_3$ can be written as $x = (x_0 + i\,x_1) + (x_2 + i\,x_3)\,j = \{x_0 + i\,x_1, x_2 + i\,x_3\}$.

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
ComplexToQuaternion[1 + 2 I, 3 + 4 I]
```

Out[2]=
```
Quaternion[1, 2, 3, 4]
```

SEE ALSO

QuaternionToComplexMatrix • QuaternionToComplex • QuaternionToMatrixR • QuaternionToMatrixL

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

DiracL

`DiracL[f]`
applies the Dirac operator from the left to f.

MORE INFORMATION

- To use `DiracL`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`
- The function accepts as argument objects of the form `Quaternion`.
- DiracL[f] = $(I\,\partial_{x1} + J\,\partial_{x2}\,K\,\partial_{x3})\,(f_0 + I\,f_1 + J\,f_2 + K\,f_3)$

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
$CoordinatesList
```

Out[2]=
```
{X0, X1, X2, X3}
```

In[3]:=
```
DiracL[Quaternion[X0^2 - X1^2 - X2^2 - X3^2, 2 X0 X1, 2 X0 X2, 2 X0 X3]]
```

Out[4]=
```
Quaternion[-6 X0, -2 X1, -2 X2, -2 X3]
```

SEE ALSO

CauchyRiemannL • **CauchyRiemannR** • **DiracL**

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

DiracR

`DiracR[f]`
applies the Dirac operator from the right to f.

MORE INFORMATION

- To use `DiracR`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`
- The function accepts as argument objects of the form `Quaternion`.
- $\mathrm{DiracR}[f] = (f_0 + I f_1 + J f_2 + K f_3)\left(I\,\partial_{x1} + J\,\partial_{x_2} + K\,\partial_{x_3}\right).$

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
$CoordinatesList
```

Out[2]=
```
{X0, X1, X2, X3}
```

In[3]:=
```
DiracR[Quaternion[X0^2 - X1^2 - X2^2 - X3^2, 2 X0 X1, 2 X0 X2, 2 X0 X3]]
```

Out[4]=
```
Quaternion[-6 X0, -2 X1, -2 X2, -2 X3]
```

SEE ALSO

CauchyRiemannL • CauchyRiemannR • DiracR

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Laplace

`Laplace[f]`
applies the Laplace operator to f.

MORE INFORMATION

- To use `Laplace`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]
- The function accepts as argument objects of the form `Quaternion`.
- $\text{Laplace}[f] = \left(\partial_{x0}^2 + \partial_{x1}^2 + \partial_{x2}^2 + \partial_{x3}^2\right) f$

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= $CoordinatesList

Out[2]= {X0, X1, X2, X3}

In[3]:= Laplace[Quaternion[X0^3 - X1^2 - X2^2 - X3^2, 2 X0 ^4 X1,  2 X0 X2, 2 X0 X3]]

Out[4]= Quaternion[-6 + 6 X0, 24 X0^2 X1, 0, 0]
```

SEE ALSO

CauchyRiemannL • CauchyRiemannR • DiracL • DiracR

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

LeftMonogenicQ

LeftMonogenicQ[f]
gives True if f is left monogenic, i.e. CauchyRiemannL[f]=0.

MORE INFORMATION

- To use `LeftMonogenicQ`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`.
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion`, `Paravector` and `ComplexLike`.

SEE ALSO

CauchyRiemannL • CauchyRiemannR • MonogenicQ • RightMonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

MonogenicQ

MonogenicQ[f]
gives True if f is monogenic, i.e. CauchyRiemannL[f]=CauchyRiemannR[f]=0.

MORE INFORMATION

- To use `MonogenicQ`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`.
- The function accepts as arguments objects of the form `Quaternion`, `Paravector` and `ComplexLike`.

EXAMPLES

Basic Examples (4)

In[1]:= `Needs["QuaternionAnalysis`"]`

SetCoordinates::valid : The coordinates system is set to {X0, X1, X2, X3}.

In[2]:= `MonogenicQ[Quaternion[X1 X2 X3 - 1, 1 - X0 X2 X3, 1 - X0 X1 X3, 1 - X0 X1 X2]]`

Out[3]= `True`

In order to use other variables, one need to change the coordinates system

In[1]:= `SetCoordinates[a, b, c]`

SetCoordinates::valid : The coordinates system is set to {a, b, c}.

In[1]:= MonogenicQ$\left[\text{Paravector}\left[a^2 - \frac{b^2}{2} - \frac{c^2}{2}, a\,b, a\,c\right]\right]$

Out[2]= `True`

The default variables for the ComplexLike objects are the first element of $CoordinatesList and R

In[1]:= `$CoordinatesList`

Out[1]= `{a, b, c}`

In[2]:= MonogenicQ$\left[\text{ComplexLike}\left[-\frac{R^2}{2} + a^2, R\,a\right]\right]$

Out[2]= `True`

One can use other variables, as far as they are explicitly mentioned as arguments of MonogenicQ

In[1]:= MonogenicQ$\left[\text{ComplexLike}\left[-\frac{r^2}{2} + x0^2, r\,x0\right]\right]$

Out[1]= `False`

In[2]:= MonogenicQ$\left[\text{ComplexLike}\left[-\frac{r^2}{2} + x0^2, r\,x0\right], x0, r\right]$

Out[2]= `True`

Possible Issues (1)

For ComplexLike objects, the use of variables other than X0 and R may require additional assumptions

In[1]:= `p = Pk[3, 2, ComplexLike[X, Y]]`

Out[1]= ComplexLike$\left[-\frac{3}{2}\,\text{Im}\left[X^2\,Y\right]+\frac{3\,\text{Im}\left[Y^3\right]}{8}+\text{Re}\left[X^3-\frac{3\,X\,Y^2}{2}\right],\ \text{Im}\left[X^3-\frac{3\,X\,Y^2}{2}\right]+\frac{3}{2}\,\text{Re}\left[X^2\,Y\right]-\frac{3\,\text{Re}\left[Y^3\right]}{8}\right]$

In[2]:= `$Assumptions = $Assumptions && Y ≥ 0 && X ∈ Reals;`

In[3]:= `p = Pk[3, 2, ComplexLike[X, Y]]`

Out[3]= ComplexLike$\left[X^3-\frac{3\,X\,Y^2}{2},\ -\frac{3}{8}\,Y\left(-4\,X^2+Y^2\right)\right]$

In[4]:= `MonogenicQ[%, X, Y]`

Out[4]= True

SEE ALSO

CauchyRiemannL • CauchyRiemannR • LeftMonogenicQ • RightMonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Paravector

`Paravector[x0, x1, ..., xn]`

represents the paravector x0 + x1 e_1 +...+ xn e_n, where $\{e_1, \ldots, e_n\}$ is an orthonormal basis of the Euclidean vector space $\mathbb{R}^n$ with a product according to the multiplication rules $e_k e_l = -e_k e_l$, if $k \neq l$ and $e_k^2 = -1$.

MORE INFORMATION

- To use `Paravector`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]

EXAMPLES

Basic Examples (2)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

The `Paravector` object:

In[2]:=
```
SetCoordinates[5]
```

Coordinates::valid : The coordinates system is set to {X0, X1, X2, X3, X4}.

In[2]:=
```
Paravector[1, 2, 3, 4, 5] // TraditionalForm
```

Out[2]= $e_0 + 2\, e_1 + 3\, e_2 + 4\, e_3 + 5\, e_4$

Operations on `Paravector` objects:

In[1]:=
```
Paravector[1, 2, 3, 4, 5] + 2 Paravector[5, 4, 3, 2, 1]
```

Out[1]= `Paravector[11, 10, 9, 8, 7]`

In[1]:=
```
Conjugate[Paravector[1, 2, 3, 4, 5]]
```

Out[1]= `Paravector[1, -2, -3, -4, -5]`

Possible Issues (1)

The product of two paravectors is not, in general, a paravector. Hence, the multiplication is not extended for this class of objects.

In[1]:=
```
Paravector[1, 2, 3, 4, 5] ** Paravector[5, 4, 3, 2, 1]
```

Out[1]= `Paravector[1, 2, 3, 4, 5] ** Paravector[5, 4, 3, 2, 1]`

SEE ALSO

SetCoordinates • $CoordinatesList • $Dim • Quaternion • ComplexLike

TUTORIALS

- Quaternion Analysis Package

- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Pk

Pk[k,n,x]
gives the monogenic polynomial of degree k, generalizing the complex powers.

MORE INFORMATION

- To use Pk, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`.
- Mathematical function suitable for both symbolic and numerical manipulation.
- If x is a quaternion (n=3) or a 3D-paravector (n=2),

$$P_k^n(x) = \sum_{s=0}^{k} T_s^k \, x^{k-s} \, \overline{x}^s.$$

- If x=ComplexLike[a,b]=a+ωb, Pk can be written in the more convenient form

$$P_k^n(x) = \sum_{s=0}^{k} \binom{k}{s} c_s(n) \, x_0^{k-s} \, \underline{x}^s$$

EXAMPLES

Basic Examples (3)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
SetCoordinates[x0, x1, x2, x3]
x = Quaternion[x0, x1, x2, x3];
TableForm[Table[{StringJoin[{"Pk[", ToString[k], ",x]="}], Pk[k, x]}, {k, 0, 3}]]
```

SetCoordinates::valid : The coordinates system is set to {x0, x1, x2, x3}.

Out[2]//TableForm=

Pk[0,x]=	Quaternion[1, 0, 0, 0]
Pk[1,x]=	$\text{Quaternion}\left[x0, \frac{x1}{3}, \frac{x2}{3}, \frac{x3}{3}\right]$
Pk[2,x]=	$\text{Quaternion}\left[x0^2 - \frac{x1^2}{3} - \frac{x2^2}{3} - \frac{x3^2}{3}, \frac{2\,x0\,x1}{3}, \frac{2\,x0\,x2}{3}, \frac{2\,x0\,x3}{3}\right]$
Pk[3,x]=	$\text{Quaternion}\left[x0\,(x0^2 - x1^2 - x2^2 - x3^2), -\frac{1}{5}\,x1\,(-5\,x0^2 + x1^2 + x2^2 + x3^2), -\frac{1}{5}\,x2\,(-5\,x0^2 + x1^2 + x2^2 + x3\right.$

In[1]:=
```
SetCoordinates[x0, x1, x2]; x = Paravector[x0, x1, x2];
TableForm[Table[{StringJoin[{"Pk[", ToString[k], ",x]="}], Pk[k, x]}, {k, 0, 3}]]
```

SetCoordinates::valid : The coordinates system is set to {x0, x1, x2}.

Out[1]//TableForm=

Pk[0,x]=	Paravector[1, 0, 0]
Pk[1,x]=	$\text{Paravector}\left[x0, \frac{x1}{2}, \frac{x2}{2}\right]$
Pk[2,x]=	$\text{Paravector}\left[x0^2 - \frac{x1^2}{2} - \frac{x2^2}{2}, x0\,x1, x0\,x2\right]$
Pk[3,x]=	$\text{Paravector}\left[x0^3 - \frac{3}{2}\,x0\,(x1^2 + x2^2), -\frac{3}{8}\,x1\,(-4\,x0^2 + x1^2 + x2^2), -\frac{3}{8}\,x2\,(-4\,x0^2 + x1^2 + x2^2)\right]$

In[1]:=
```
TableForm[Table[{StringJoin[{"Pk[", ToString[k], ",4,x]="}], Pk[k, 4, ComplexLike[X0, R]]}, {k, 0, 3}]]
```

Out[1]//TableForm=

Pk[0,4,x]=	ComplexLike[1, 0]
Pk[1,4,x]=	ComplexLike$\left[X0, \frac{R}{4}\right]$
Pk[2,4,x]=	ComplexLike$\left[-\frac{R^2}{4} + X0^2, \frac{R\,X0}{2}\right]$
Pk[3,4,x]=	ComplexLike$\left[-\frac{3\,R^2\,X0}{4} + X0^3, -\frac{1}{8}\,R\left(R^2 - 6\,X0^2\right)\right]$

SEE ALSO

Tks • Ck • MonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

PolarForm

`PolarForm[`*expr*`]`
gives de polar form of *expr*.

MORE INFORMATION

- To use `PolarForm`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"`] .
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion` and `Paravector`.

EXAMPLES

Basic Examples (1)

In[1]:= `Needs["QuaternionAnalysis`"]`

In[2]:= `PolarForm[Quaternion[1, 2, 3, 4]]`

Out[2]= $\left\{\sqrt{30}, \text{ArcCot}\left[\frac{1}{\sqrt{29}}\right]\right\}$

In[3]:= `PolarForm[Paravector[1, 2, 3, 4, 5]]`

Out[3]= $\left\{\sqrt{55}, \text{ArcCot}\left[\frac{1}{3\sqrt{6}}\right]\right\}$

SEE ALSO

Quaternion • Paravector • ComplexLike • ToComplexLike

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

PureQuaternionQ

`PureQuaternionQ[expr]`
gives True if *expr* is a pure quaternions and False otherwise.

MORE INFORMATION

- To use `PureQuaternionQ`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]`
- Mathematical function suitable for both symbolic and numerical manipulation.

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= q = Quaternion[1, 2, 3, 4];
        PureQuaternionQ[q]
Out[3]= False

In[4]:= PureQuaternionQ[Vec[q]]
Out[4]= True

In[5]:= PureQuaternionQ[Quaternion[a, b, c, d]]
Out[5]= False
```

SEE ALSO

Quaternion • **Vec**

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

QPower

`QPower[`*expr*, *n*`]`
a recursive definition of *exp^n*

MORE INFORMATION

- To use `QPower`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]`.
- Mathematical function suitable for both symbolic and numerical manipulation. For symbolic computations we recommend de use of QPower rather than Power.
- The function accepts as arguments objects of the form `Quaternion` and `ComplexLike`.

EXAMPLES

Basic Examples (2)

In[1]:= `Needs["QuaternionAnalysis`"]`

In[2]:= `QPower[Quaternion[1, 2, 3, 4], 2]`

Out[2]= `Quaternion[-28, 4, 6, 8]`

In[3]:= `Power[Quaternion[1, 2, 3, 4], 2]`

Out[3]= `Quaternion[-28, 4, 6, 8]`

In[1]:= `QPower[Quaternion[a, b, c, d], 2]`

Out[1]= $\text{Quaternion}\left[a^2-b^2-c^2-d^2,\ 2\,a\,b,\ 2\,a\,c,\ 2\,a\,d\right]$

In[2]:= `Power[Quaternion[a, b, c, d], 2]`

Out[2]= $$\text{Quaternion}\left[a^2-b^2-c^2-d^2,\ 2\,a\,b,\ 2\,a\sqrt{\frac{c^2+d^2}{b^2+c^2+d^2}}\ \sqrt{b^2+c^2+d^2}\ \text{Cos}\left[\text{ArcCos}\left[\frac{c}{\sqrt{\frac{c^2+d^2}{b^2+c^2+d^2}}\ \sqrt{b^2+c^2+d^2}}\right]\text{Sign}[d]\right],\right.$$
$$\left.2\,a\sqrt{\frac{c^2+d^2}{b^2+c^2+d^2}}\ \sqrt{b^2+c^2+d^2}\ \text{Sin}\left[\text{ArcCos}\left[\frac{c}{\sqrt{\frac{c^2+d^2}{b^2+c^2+d^2}}\ \sqrt{b^2+c^2+d^2}}\right]\text{Sign}[d]\right]\right]$$

SEE ALSO

Power • NonCommutativeMultiply

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

QuaternionToComplex

QuaternionToComplex[x]
gives a complex representation of the quaternion x.

MORE INFORMATION

- To use `QuaternionToComplex`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]
- Mathematical function suitable for both symbolic and numerical manipulation.
- Any quaternion $x = x_0 + i\,x_1 + j\,x_2 + k\,x_3$ can be written as $x = (x_0 + i\,x_1) + (x_2 + i\,x_3)\,j = \{x_0 + i\,x_1, x_2 + i\,x_3\}$.

EXAMPLES

Basic Examples (2)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= QuaternionToComplex[Quaternion[1, 2, 3, 4]]

Out[2]= {1 + 2 i, 3 + 4 i}
```

```
In[1]:= QuaternionToComplex[Quaternion[a, b, c, d]]

Out[1]= {a + i b, c + i d}
```

SEE ALSO

QuaternionToComplexMatrix • ComplexToQuaternion • QuaternionToMatrixR • QuaternionToMatrixL

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

QuaternionToComplexMatrix

QuaternionToComplexMatrix[x]
    gives a complex representation matrix of the quaternion x.

MORE INFORMATION

- To use `QuaternionToComplexMatrix`, you first need to load the `Quaternion Analysis Package` using Needs["`Quaternion Analysis`"]
- Mathematical function suitable for both symbolic and numerical manipulation.
- Any quaternion $x = x_0 + i x_1 + j x_2 + k x_3$ can be written as $\begin{pmatrix} x_0 + i x_1 & x_2 + i x_3 \\ -x_2 + i x_3 & x_0 - i x_1 \end{pmatrix}$.

EXAMPLES

Basic Examples (2)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
x = Quaternion[1, 2, 3, 4];
y = Quaternion[4, 3, 2, 1];
```

In[3]:=
```
QuaternionToComplexMatrix[x] // MatrixForm
```

Out[3]//MatrixForm=

$$\begin{pmatrix} 1+2i & 3+4i \\ -3+4i & 1-2i \end{pmatrix}$$

In[4]:=
```
QuaternionToComplexMatrix[x].QuaternionToComplexMatrix[y] // MatrixForm
```

Out[4]//MatrixForm=

$$\begin{pmatrix} -12+6i & 24+12i \\ -24+12i & -12-6i \end{pmatrix}$$

In[5]:=
```
QuaternionToComplexMatrix[x ** y] // MatrixForm
```

Out[5]//MatrixForm=

$$\begin{pmatrix} -12+6i & 24+12i \\ -24+12i & -12-6i \end{pmatrix}$$

In[1]:=
```
QuaternionToComplexMatrix[Quaternion[x0, x1, x2, x3]] // MatrixForm
```

Out[1]//MatrixForm=

$$\begin{pmatrix} x0 + i\,x1 & x2 + i\,x3 \\ -x2 + i\,x3 & x0 - i\,x1 \end{pmatrix}$$

SEE ALSO

QuaternionToComplex • ComplexToQuaternion • QuaternionToMatrixR • QuaternionToMatrixL

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

QuaternionToMatrixL

QuaternionToMatrixL[x]
    gives a real left representation matrix of the quaternion x.

MORE INFORMATION

- To use `QuaternionToMatrixL`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]
- Mathematical function suitable for both symbolic and numerical manipulation.
- The left representation of the quaternion $x = x_0 + i\,x_1 + j\,x_2 + k\,x_3$ is

$$L_x = \begin{pmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & -x_3 & x_2 \\ x_2 & x_3 & x_0 & -x_1 \\ x_3 & -x_2 & x_1 & x_0 \end{pmatrix}$$

EXAMPLES

Basic Examples (2)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= x = Quaternion[1, 2, 3, 4];
        y = Quaternion[4, 3, 2, 1];

In[3]:= (Lx = QuaternionToMatrixL[x]) // MatrixForm
```

Out[3]//MatrixForm=

$$\begin{pmatrix} 1 & -2 & -3 & -4 \\ 2 & 1 & -4 & 3 \\ 3 & 4 & 1 & -2 \\ 4 & -3 & 2 & 1 \end{pmatrix}$$

```
In[4]:= Quaternion @@ (Lx.List @@ y)

Out[4]= Quaternion[-12, 6, 24, 12]

In[5]:= x ** y

Out[5]= Quaternion[-12, 6, 24, 12]
```

```
In[1]:= x = Quaternion[x0, x1, x2, x3];
        y = Quaternion[y0, y1, y2, y3];

In[2]:= (Lx = QuaternionToMatrixL[x]) // MatrixForm
```

Out[2]//MatrixForm=

$$\begin{pmatrix} x0 & -x1 & -x2 & -x3 \\ x1 & x0 & -x3 & x2 \\ x2 & x3 & x0 & -x1 \\ x3 & -x2 & x1 & x0 \end{pmatrix}$$

```
In[3]:= Lx.List @@ y

Out[3]= {x0 y0 - x1 y1 - x2 y2 - x3 y3, x1 y0 + x0 y1 - x3 y2 + x2 y3, x2 y0 + x3 y1 + x0 y2 - x1 y3, x3 y0 - x2 y1 + x1 y2 + x0 y3}
```

In[4]:= `List @@ (x ** y)`

Out[4]= {x0 y0 − x1 y1 − x2 y2 − x3 y3, x1 y0 + x0 y1 − x3 y2 + x2 y3, x2 y0 + x3 y1 + x0 y2 − x1 y3, x3 y0 − x2 y1 + x1 y2 + x0 y3}

SEE ALSO

QuaternionToMatrixR • QuaternionToComplexMatrix • ComplexToQuaternion • QuaternionToComplex

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

QuaternionToMatrixR

QuaternionToMatrixL[x]
gives a real left representation matrix of the quaternion x.

MORE INFORMATION

- To use `QuaternionToMatrixR`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]
- Mathematical function suitable for both symbolic and numerical manipulation.
- The right representation of the quaternion $x = x_0 + i\, x_1 + j\, x_2 + k\, x_3$ is

$$R_x = \begin{pmatrix} x_0 & -x_1 & -x_2 & -x_3 \\ x_1 & x_0 & x_3 & -x_2 \\ x_2 & -x_3 & x_0 & x_1 \\ x_3 & x_2 & -x_1 & x_0 \end{pmatrix}$$

EXAMPLES

Basic Examples (2)

```
In[1]:= Needs["QuaternionAnalysis`"]

In[2]:= x = Quaternion[1, 2, 3, 4];
        y = Quaternion[4, 3, 2, 1];

In[3]:= (Rx = QuaternionToMatrixR[x]) // MatrixForm
```

Out[3]//MatrixForm=

$$\begin{pmatrix} 1 & -2 & -3 & -4 \\ 2 & 1 & 4 & -3 \\ 3 & -4 & 1 & 2 \\ 4 & 3 & -2 & 1 \end{pmatrix}$$

```
In[4]:= Quaternion @@ (Rx . List @@ y)

Out[4]= Quaternion[-12, 16, 4, 22]

In[5]:= y ** x

Out[5]= Quaternion[-12, 16, 4, 22]
```

```
In[1]:= x = Quaternion[x0, x1, x2, x3];
        y = Quaternion[y0, y1, y2, y3];

In[2]:= (Rx = QuaternionToMatrixR[x]) // MatrixForm
```

Out[2]//MatrixForm=

$$\begin{pmatrix} x0 & -x1 & -x2 & -x3 \\ x1 & x0 & x3 & -x2 \\ x2 & -x3 & x0 & x1 \\ x3 & x2 & -x1 & x0 \end{pmatrix}$$

```
In[3]:= Rx . List @@ y

Out[3]= {x0 y0 - x1 y1 - x2 y2 - x3 y3, x1 y0 + x0 y1 + x3 y2 - x2 y3, x2 y0 - x3 y1 + x0 y2 + x1 y3, x3 y0 + x2 y1 - x1 y2 + x0 y3}
```

In[4]:= `List @@ (y ** x)`

Out[4]= {x0 y0 − x1 y1 − x2 y2 − x3 y3, x1 y0 + x0 y1 + x3 y2 − x2 y3, x2 y0 − x3 y1 + x0 y2 + x1 y3, x3 y0 + x2 y1 − x1 y2 + x0 y3}

SEE ALSO

QuaternionToMatrixL • QuaternionToComplexMatrix • ComplexToQuaternion • QuaternionToComplex

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

RightMonogenicQ

RightMonogenicQ[f]
gives True if f is left monogenic, i.e. CauchyRiemannL[f]=0.

MORE INFORMATION

- To use `RightMonogenicQ`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]`.
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion`, `Paravector` and `ComplexLike`.

SEE ALSO

CauchyRiemannL • CauchyRiemannR • MonogenicQ • LeftMonogenicQ

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

SetCoordinates

`SetCoordinates[CoordinatesSequence]`
sets the default coordinates to be CoordinatesSequence.

`SetCoordinates[a, b, c]`
sets the default coordinates to be a, b and c.

`SetCoordinates[5]`
sets the default coordinates to be X0, X1, X2, X3 and X4.

SEE ALSO

$CoordinatesList • $Dim • MonogenicQ • Paravector • ComplexLike

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

SymmetricProduct

SymmetricProduct[x, y]
gives de symmetric product of the quaternions x and y.

MORE INFORMATION

- To use SymmetricPower, you first need to load the Quaternion Analysis Package using Needs["QuaternionAnalysis`"].
- Mathematical function suitable for both symbolic and numerical manipulation.
- SymmetricProduct[x, y] = $\frac{x**y+y**x}{2}$

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]
```

A commutative product

```
In[2]:= q1 = Quaternion[1, 2, 3, 4];
        q2 = Quaternion[1, 1, -1, 1];
        q3 = Quaternion[1, 0, 0, 1];

In[3]:= SymmetricProduct[q1, q2]

Out[3]= Quaternion[-2, 3, 2, 5]

In[4]:= SymmetricProduct[q2, q1]

Out[4]= Quaternion[-2, 3, 2, 5]
```

but not associative

```
In[5]:= SymmetricProduct[SymmetricProduct[q1, q2], q3]

Out[5]= Quaternion[-7, 3, 2, 3]

In[6]:= SymmetricProduct[q1, SymmetricProduct[q2, q3]]

Out[6]= Quaternion[-7, 1, -1, 2]
```

SEE ALSO

NonCommutativeMultiply

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Tks

`Tks[k, s, n]`
gives the (k,s) element of a triangle associated with `Pk`.

`Tks[k, s] = Tks[k, s, $Dim - 1]`.

MORE INFORMATION

- To use `Tks`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]`.
- Mathematical function suitable for both symbolic and numerical manipulation.
- T_s^k are the numbers

$$T_s^k = \frac{k!}{(n)_k} \frac{\left(\frac{n+1}{2}\right)_{k-s} \left(\frac{n-1}{2}\right)_s}{(k-s)!\, s!}$$

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

In[2]:=
```
TableForm[Table[TableForm[Table[Tks[k, s, m], {k, 0, 4 }, {s, 0, k }]], {m, {2, 3, 5}}],
 TableDirections -> Row, TableHeadings → {{" Tks(2)", " Tks(3)", " Tks(5)" }}]
```

Out[2]//TableForm=

Tks(2)					Tks(3)					Tks(5)				
1					1					1				
$\frac{3}{4}$	$\frac{1}{4}$				$\frac{2}{3}$	$\frac{1}{3}$				$\frac{3}{5}$	$\frac{2}{5}$			
$\frac{5}{8}$	$\frac{1}{4}$	$\frac{1}{8}$			$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{6}$			$\frac{2}{5}$	$\frac{2}{5}$	$\frac{1}{5}$		
$\frac{35}{64}$	$\frac{15}{64}$	$\frac{9}{64}$	$\frac{5}{64}$		$\frac{2}{5}$	$\frac{3}{10}$	$\frac{1}{5}$	$\frac{1}{10}$		$\frac{2}{7}$	$\frac{12}{35}$	$\frac{9}{35}$	$\frac{4}{35}$	
$\frac{63}{128}$	$\frac{7}{32}$	$\frac{9}{64}$	$\frac{3}{32}$	$\frac{7}{128}$	$\frac{1}{3}$	$\frac{4}{15}$	$\frac{1}{5}$	$\frac{2}{15}$	$\frac{1}{15}$	$\frac{3}{14}$	$\frac{2}{7}$	$\frac{9}{35}$	$\frac{6}{35}$	$\frac{1}{14}$

SEE ALSO

Pk • Ck • Pochhammer

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

ToComplexLike

`ToComplexLike[`*expr*`]`
gives *expr* in the complex-like form.

MORE INFORMATION

- To use `ToComplexLike`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]`
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion` and `Paravector`.

EXAMPLES

Basic Examples (1)

```
In[1]:= Needs["QuaternionAnalysis`"]
```

`Quaternions` or `Paravector` objects:

```
In[2]:= ToComplexLike[Quaternion[a, b, c, d]]
```

Out[3]= ComplexLike$\left[a, \sqrt{b^2 + c^2 + d^2}\right]$

```
In[4]:= x = Paravector[1, 2, 3];
```

```
In[5]:= ToComplexLike[x]
```

Out[5]= ComplexLike$\left[1, \sqrt{13}\right]$

```
In[6]:= Re[x] + W[x] AbsVec[x]
Out[6]= Paravector[1, 2, 3]
```

SEE ALSO

Quaternion • Paravector • ComplexLike • W • AbsVec

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

Vec

`Vec[expr]`
gives the vector part of *expr*.

MORE INFORMATION

- To use `Vec`, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis``"]` .
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion`, `Paravector` and `ComplexLike`.

EXAMPLES

Basic Examples (3)

```
In[1]:= Needs["QuaternionAnalysis`"]
```

Quaternion numbers:

```
In[2]:= Vec[Quaternion[1, 2, 3, 4]]
Out[2]= Quaternion[0, 2, 3, 4]
```

Paravector numbers:

```
In[1]:= Vec[Paravector[1, 2, 3]]
Out[1]= Paravector[0, 2, 3]
```

```
In[2]:= Vec[Paravector[a, b, c, d, e]] // TraditionalForm
```

Out[2]= $b\,e_1 + c\,e_2 + d\,e_3 + e\,e_4$

ComplexLike representation:

```
In[1]:= Vec[ComplexLike[1, 2]]
Out[1]= ComplexLike[0, 2]
```

```
In[2]:= % // TraditionalForm
Out[2]= 2 w
```

SEE ALSO

Quaternion • Paravector • ComplexLike

TUTORIALS

- Quaternion Analysis Package

- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

W

`W[`*expr*`]`
gives the sign of the vector part of *expr*.

MORE INFORMATION

- To use W, you first need to load the `Quaternion Analysis Package` using Needs["`QuaternionAnalysis`"]
- Mathematical function suitable for both symbolic and numerical manipulation.
- The function accepts as arguments objects of the form `Quaternion` and `Paravector`.

EXAMPLES

Basic Examples (1)

In[1]:=
```
Needs["QuaternionAnalysis`"]
```

`Quaternions` or `Paravector` objects:

In[2]:=
```
W[Quaternion[a, b, c, d]]
```

Out[3]= $\text{Quaternion}\left[0, \frac{b}{\sqrt{b^2+c^2+d^2}}, \frac{c}{\sqrt{b^2+c^2+d^2}}, \frac{d}{\sqrt{b^2+c^2+d^2}}\right]$

In[4]:=
```
x = Paravector[1, 4, 3];
W[x]
```

Out[4]= $\text{Paravector}\left[0, \frac{4}{5}, \frac{3}{5}\right]$

In[5]:=
```
Re[x] + W[x] AbsVec[x]
```

Out[5]= `Paravector[1, 4, 3]`

SEE ALSO

Quaternion • **Paravector** • **ComplexLike** • **ToComplexLike** • **AbsVec**

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

$CoordinatesList

`$CoordinatesList`

is the list of variables. {X0,X1,X2,X3} is the default coordinates list.

SEE ALSO

SetCoordinates • $Dim

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis

$Dim

`$Dim`

is the dimension of the space. By default the space dimension is 4. The `$Dim` value is automatically adjusted by the `SetCoordinates` function.

SEE ALSO

SetCoordinates • $CoordinatesList

TUTORIALS

- Quaternion Analysis Package
- Quaternions Package

MORE ABOUT

- QuaternionAnalysis