qPoly RELEASE 24.05
USER MANUAL

PRZEMYSŁAW KOPROWSKI

This manual describes the MAGMA [3] package qPoly for working with quaternionic polynomials. The main purpose of the library is to extend MAGMA functionality allowing it to factor and find roots of quaternionic polynomials over number fields. The software can be downloaded for free from the author's webpage at `http://www.pkoprowski.eu/qpoly`. The package is released under standard MIT license. The full statement of the license is given at the end of this manual.

The library was developed on version 2.28 of MAGMA. The author has not tested it on other versions (yet), but will appropriate any reports on how it works in different setups. Questions and comments should be sent via email to `przemyslaw.koprowski@us.edu.pl`.

1. Quick start guide

We present here a quick glimpse of the use of the package. Full description of the functions used in this tutorial can be found in the following sections. We assume that the reader is at least mildly familiar with MAGMA. Otherwise we suggest to consult [6] prior to reading this manual.

All the examples presented in this manual can be found in the file `examples.magma` in the root directory of qPoly. First of all, we must load qPoly (see Section 2):

```
> AttachSpec("path_to_qpoly/qpoly.spec");
```

Construct a quaternion algebra $\mathscr{A} := \left(\frac{-1,-1}{\mathbb{Q}}\right)$ over the rationals and the ring of quaternionic polynomials:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
```

We can now define a quaternionic polynomial $\wp = (x-\hbar)(x-j)(x-i)$:

```
> p := (x-kk)*(x-jj)*(x-ii); p;
x^3 + (-ii - jj - kk)*x^2 + (-ii + jj - kk)*x + -1
```

The library qPoly extends the command `Factorization`, so that Magma can factor quaternionic polynomials:

```
> Factorization(p);
[
    <x + -kk, 1>,
    <x + -jj, 1>,
    <x + -ii, 1>
]
```

We can check whether $\wp$ has any roots in $\mathscr{A}$:

Date: April 15, 2025.

```
> HasRoot(p);
true ii
```

Indeed, i is a root of $\wp$:

```
> Evaluate(p, ii);
0
```

2. Package installation and loading

The package can be downloaded for free from the author's web page

`http://www.pkoprowski.eu/qpoly`

The package comes in form of a zip archive that must be extracted an put on disc in a place where Magma is able to find it. To test the installation go to the directory where $q\mathcal{P}oly$ resides and execute

```
magma tests.magma
```

If it prints "All tests passed" this means that the installation is correct.

The package is loaded into Magma by attaching its spec file:

```
> AttachSpec("path_to_qpoly/qpoly.spec");
```

One can also add this command to Magma startup file (specified by the environment variable `MAGMA_USER_SPEC`) to auto-load the package when Magma starts. For details about Magma startup environment consult [5, Sections 2.3.9 and 4.3].

3. List of functions

Below we list all the functions provided by $q\mathcal{P}oly$. When possible, the functions implemented by this library have the same names as functions already existing in Magma, extending the built-in function to quaternionic polynomials. Beware that most functions require the underlying quaternion algebra to be a division algebra.

3.1. **Predicates.**

`IsCenterfree(p)`

Let $\wp \in \mathscr{A}[x]$ be a polynomial with coefficients in a division quaternion algebra. This function returns `true` if and only if $\wp$ is center-free, that is it is not divisible by any non-constant polynomial with coefficients in the base field of $\mathscr{A}$, otherwise the function outputs false.

`IsCentral(p)`

Given a quaternionic polynomial $\wp \in \mathscr{A}[x]$ this function outputs `true` if and only if $\wp$ is in the center of $\mathscr{A}[x]$.

`IsCongruent(a, b)`

`IsCongruent(p, q)`

`CongruenceFactor` BoolElt *Default:* `false`

Let $\mathscr{A}$ be a division quaternion algebra. Two quaternions $a, b \in \mathscr{A}$ are called *congruent* (or conjugate or similar) if there is a nonzero quaternion c such that $ac = cb$. Likewise, two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ are said to be congruent, if there is a polynomial $\mathfrak{r} \in \mathscr{A}[x]$ such that $\wp\mathfrak{r} = \mathfrak{r}\mathfrak{q}$. This function works with either quaternions or quaternionic polynomials and return `true` if the two given quaternions (respectively quaternionic polynomials) are congruent and `false` otherwise.

If the optional argument `CongruenceFactor` is set to `true` (by default it is `false`) and the arguments are congruent, the function will return as a second value the quaternion c such that $ac = cb$ (resp. the quaternionic polynomial $r \in \mathscr{A}[x]$ such that $\wp r = r q$).

Example 1. Define a quadratic number field $K = \mathbb{Q}(\sqrt{3})$ and a division quaternion algebra $\mathscr{A} = \left(\frac{-1,-3}{K}\right)$:

```
> K<sqrt3> := QuadraticField( 3 );
> A<ii,jj,kk> := QuaternionAlgebra<K | -1, -3>;
```

Then quaternions $\sqrt{3} \cdot i$ and j are congruent:

```
> IsCongruent(sqrt3*ii, jj);
true
```

`IsIrreducible( p )`

Assume that $\mathscr{A}$ is a division quaternion algebra. Given a quaternionic polynomial $\wp \in \mathscr{A}[x]$, this function return `true` if $\wp$ is irreducible in $\mathscr{A}[x]$ and `false` otherwise.

Example 2. Use the quaternion algebra $\mathscr{A} = \left(\frac{-1,-1}{\mathbb{Q}}\right)$:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
```

Consider a quaternionic polynomial $\wp = x^3 - (i - j) \cdot x^2 - 2k \cdot x - j$:

```
> p := x^3 - (ii - jj)*x^2 - 2*kk*x - jj; p;
x^3 + (-ii + jj)*x^2 + -2*kk*x + -jj
```

Check whether $\wp$ is irreducible:

```
> IsIrreducible(p);
false
```

Indeed, it factors into a product of 3 linear terms (see below for the description of the `Factorization` command):

```
> L := Factorization(p); L;
[
    <x + -ii, 1>,
    <x + 1/2 - 1/2*ii + 1/2*jj + 1/2*kk, 1>,
    <x + -1/2 + 1/2*ii + 1/2*jj - 1/2*kk, 1>
]
> &* [ q[1] : q in L ] eq p;
true
```

Example 3. In this example we show that a polynomial irreducible in $K[x]$ can become reducible in $\mathscr{A}[x]$, where $\mathscr{A}$ is a division quaternion algebra over K.

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Qx<x> := PolynomialRing(QQ);
> Ax := PolynomialRing(A);
> p := 2*x^4 - 6*x^3 + 20*x^2 - 10*x + 14; p;
2*x^4 - 6*x^3 + 20*x^2 - 10*x + 14
```

Check that p is irreducible in $\mathbb{Q}[x]$:

```
> IsIrreducible( p );
true
```

Nevertheless, it is reducible in $\mathscr{A}[x]$:

```
> IsIrreducible( Ax!p );
false
```

`IsSimilar(p, q)`

`Isomorphism` BoolElt *Default:* `false`

Let $\mathscr{A}$ be a division quaternion algebra. Two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ are called *similar* if the left $\mathscr{A}[x]$-modules $\mathscr{A}[x]/(\mathscr{A}[x] \cdot \wp)$ and $\mathscr{A}[x]/(\mathscr{A}[x] \cdot \mathfrak{q})$ are isomorphic.

This function verifies whether two monic quaternionic polynomials are similar. If they are similar and the optional parameter `Isomorphism` is set to `true`, then the function outputs the corresponding isomorphism as its second value.

Warning: At present the polynomials $\wp$ and $\mathfrak{q}$ must be monic. This is due to the fact that Magma supports construction of quotient algebras only for monic polynomials. Similarity is oblivious to multiplication by units, hence this requirement is not restrictive.

`IsSquare(a)`

`ComputeSqrt` BoolElt *Default:* `true`

Given a quaternion $a \in \mathscr{A}$, this function outputs `true` when a is a square in $\mathscr{A}$ and `false` otherwise. In addition, if the optional argument `ComputeSqrt` is `true` (which is its default value) and a is a square, then the function outputs its square root.

Warning: Unlike most other functions, `IsSquare` works just with quaternions not quaternionic polynomials.

3.2. **Basic function.**

`Conjugate(p)`

If $\wp = a_0 + a_1 x + \cdots + a_d x^d$ is a quaternionic polynomial, then the *conjugate* of $\wp$ is $\overline{\wp} = \overline{a_0} + \overline{a_1} x + \cdots + \overline{a_d} x^d$. This function returns the conjugate of a quaternionic polynomial $\wp \in \mathscr{A}[x]$.

`Coordinates(p)`

Every quaternionic polynomial $\wp \in \mathscr{A}[x]$ can be uniquely written in the form $\wp = p_0 + p_1 i + p_2 j + p_3 k$, for some polynomials $p_0, \ldots, p_3$ with coefficients in the center of $\mathscr{A}$. This function returns the tuple (p_0, p_1, p_2, p_3).

`MaximalCentralFactor( p )`

`CentralDecomposition( p )`

Let $\mathscr{A}$ be a division quaternion algebra over a field K (of characteristic $\operatorname{char} K \neq 2$). Beck's theorem (see [1, Résultat 4]) says that every quaternionic polynomial $\wp \in \mathscr{A}[x]$ has a unique decomposition of the form $\wp = c \cdot \mathfrak{q} \cdot p$, where:

- c is the leading coefficient of $\wp$;
- $\mathfrak{q}$ is a monic and center-free, that is it is not divisible by any non-constant polynomial from $K[x]$;

- p is a polynomial with coefficients in K, called the *maximal central factor* of $\wp$.

The function `MaximalCentralFactor` returns the maximal central factor of its argument, while the function `CentralDecomposition` outputs all three components: c, $\mathfrak{q}$, p.

Example 4. Take a polynomial

$$\wp = j \cdot x^5 + (-2 \cdot j - \hbar) \cdot x^3 + i \cdot x^2 + 2 \cdot \hbar \cdot x + -2 \cdot i$$

over the quaternion algebra $\left(\frac{-1,-1}{\mathbb{Q}}\right)$.

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
> P := jj*x^5 + (-2*jj - kk)*x^3 + ii*x^2 + 2*kk*x + -2*ii; P;
jj*x^5 + (-2*jj - kk)*x^3 + ii*x^2 + 2*kk*x + -2*ii
```

Find its maximal central factor:

```
> MaximalCentralFactor(P);
x^2 - 2
```

We can also extract all three components:

```
> c, q, p := CentralDecomposition(P);
> print "c =", c, "\nq =", q, "p =", p;
c = jj
q = x^3 + ii*x + kk
p = x^2 - 2
```

Verify that $c \cdot \mathfrak{q} \cdot p$ gives the original polynomial:

```
> c*q*p eq P;
true
```

Beware, however, of the order of multiplication. We are in a non-commutative ring. If we change the order, we are likely to obtain a different polynomial.

```
> p*q*c eq P;
false
```

`Norm(p)`

If $\wp \in \mathscr{A}[x]$ is a quaternionic polynomial, then the product $\wp \cdot \overline{\wp}$ of the polynomial and its conjugate is called the (reduced) *norm* of $\wp$. This function returns the (reduced) norm of the given quaternionic polynomial.

`Trace(p)`

If $\wp \in \mathscr{A}[x]$ is a quaternionic polynomial, then the sum $\wp + \overline{\wp}$ of the polynomial and its conjugate is called the (reduced) *trace* of $\wp$. This function returns the (reduced) trace of the given quaternionic polynomial.

3.3. **Euclidean algorithm.**

`EuclideanLeftDivision(p1, p2)`

Given two quaternionic polynomials $\wp_1, \wp_2 \in \mathscr{A}[x]$, this function returns the quotient and remainder of the **left** division of $\wp_1$ by $\wp_2$, i.e. polynomials $\mathfrak{q}, \mathfrak{r} \in \mathscr{A}[x]$ such that $\wp_1 = \wp_2 \cdot \mathfrak{q} + \mathfrak{r}$ and $\deg(\mathfrak{r}) < \deg(\wp_2)$.

`EuclideanRightDivision(p1, p2)`

Given two quaternionic polynomials $\wp_1, \wp_2 \in \mathscr{A}[x]$, this function returns the quotient and remainder of the **right** division of $\wp_1$ by $\wp_2$, i.e. polynomials $\mathfrak{q}, \mathfrak{r} \in \mathscr{A}[x]$ such that $\wp_1 = \mathfrak{q} \cdot \wp_2 + \mathfrak{r}$ and $\deg(\mathfrak{r}) < \deg(\wp_2)$.

This function is identical to the built-in function `Quotrem`. It is provided only for the sake of completeness.

`GreatestCommonLeftDivisor(p, q)`

`GCLD(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their greatest common **left** divisor.

`GreatestCommonRightDivisor(p, q)`

`GCRD(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their greatest common **right** divisor.

`ExtendedGreatestCommonLeftDivisor(p, q)`

`XGCLD(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their greatest common **left** divisor together with the corresponding **right** Bézout coefficients. That is, if the output of the function is $\mathfrak{r}, \mathfrak{a}, \mathfrak{b} \in \mathscr{A}[x]$, then $\mathfrak{r} = \wp\mathfrak{a} + \mathfrak{q}\mathfrak{b}$ is the greatest common left divisor of $\wp$ and $\mathfrak{q}$.

`ExtendedGreatestCommonRightDivisor(p, q)`

`XGCRD(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their greatest common **right** divisor together with the corresponding **left** Bézout coefficients. That is, if the output of the function is $\mathfrak{r}, \mathfrak{a}, \mathfrak{b} \in \mathscr{A}[x]$, then $\mathfrak{r} = \wp\mathfrak{a} + \mathfrak{q}\mathfrak{b}$ is the greatest common right divisor of $\wp$ and $\mathfrak{q}$.

`LeastCommonLeftMultiple(p, q)`

`LCLM(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their least common **left** multiple, ie. a monic quaternionic polynomial $\mathfrak{r}$ of lowest degree that is right-divisible by both $\wp$ and $\mathfrak{q}$.

`LeastCommonRightMultiple(p, q)`

`LCRM(p, q)`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ over a division quaternion algebra $\mathscr{A}$, this function returns their least common **right** multiple, ie. a monic quaternionic polynomial $\mathfrak{r}$ of lowest degree that is left-divisible by both $\wp$ and $\mathfrak{q}$.

Example 5. Define a division quaternion algebra $\mathscr{A} = \left(\frac{-1,-1}{\mathbb{Q}}\right)$:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
```

Now, let $\wp = (x - i - 2j)(x - i)$ and $\wp_1 = (x + i - j + k)(x - i)$ be two quaternionic polynomials:

```
> Ax<x> := PolynomialRing(A);
> p := (x - ii - 2*jj)*(x - ii);
> q := (x + ii - jj + kk)*(x - ii);
> print "p =", p, "q =", q;
p = x^2 + (-2*ii - 2*jj)*x + -1 - 2*kk
q = x^2 + (-jj + kk)*x + 1 - jj - kk
```

It is clear by the construction, that $x - i$ is their greatest common right divisor:

```
> GCRD(p, q);
x + -ii
```

Let us now compute their least common left multiple:

```
> r := LCLM(p, q); r;
x^3 + (-5/3*ii - 1/3*jj - 1/3*kk)*x^2
    + (3 - 2/3*ii + 2/3*jj + 2/3*kk)*x + -2/3 - 11/3*ii - jj + 1/3*kk
```

We can verify that r is right-divisible by both $\wp$ and $\wp_1$:

```
> print r mod p, r mod q;
0
0
```

3.4. **Root finding.**

`HasRoot( p )`

Let $\mathscr{A}$ be a division quaternion algebra over a number field K. Given a quaternionic polynomial $\wp \in \mathscr{A}[x]$ this function returns `true` if and only if $\wp$ has a root in $\mathscr{A}$. If the result is `true`, the function also outputs a root of $\wp$ as its second return value. This function is potentially faster than the computation of all roots of the polynomial $\wp$ using `Roots`.

Example 6. We will use the polynomial

$$\wp = x^4 - (3i + k) \cdot x^3 - (3 + j) \cdot x^2 + (i - k) \cdot x - j$$

over $\mathscr{A} := \left(\frac{-1,-1}{K}\right)$, where $K = \mathbb{Q}(\theta)$ is a cubic number generated by a root of the polynomial $x^3 + 3x - 1$:

```
> QX<X> := PolynomialRing(Rationals());
> K<theta> := NumberField( X^3 + 3*X - 1 );
> A<ii,jj,kk> := QuaternionAlgebra<K | -1, -1>;
> Ax<x> := PolynomialRing(A);
> p := x^4 - (3*ii + kk)*x^3 - (3 + jj)*x^2 + (ii - kk)*x - jj; p;
x^4 + (-3*ii - kk)*x^3 + (-3 - jj)*x^2 + (ii - kk)*x + -jj
```

Check whether $\wp$ has a root in $\mathscr{A}$:

```
> HasRoot(p);
true ii
```

We see that i is a root of $\wp$. We can verify the result evaluating $\wp$ at i:

```
> Evaluate(p, ii);
0
```

`Roots( p )`

Let $\mathscr{A}$ be a division quaternion algebra over a number field K. Given a quaternionic polynomial $\wp \in \mathscr{A}[x]$ this function returns a list of triples (a_i, m_i, n_i). The first element of each triple is a root of $\wp$ and the other two are the *isolated* and *spherical* multiplicities of this root.

Recall that a root of a quaternionic polynomial has **two** multiplicities! If $a \in \mathscr{A}$ is a root of $\wp$ of a positive spherical multiplicity, this means that every quaternion $b \in \mathscr{A}$ congruent to a is a root of $\wp$, as well. If the spherical multiplicities of all the roots of $\wp$ are null, then $\wp$ has only finitely many roots in $\mathscr{A}$. In general the sum of multiplicities satisfies the condition:

$$\sum_i (m_i + n_i) \leq \deg \wp.$$

We have an equality in the above formula if and only if $\wp$ is a product of linear factors.

Warning: The notion of root multiplicity of a quaternionic polynomial is not fuly consistent in the literature. The convention used here agrees with the one in [8, 13]. But, for example, in [11] the spherical multiplicity is defined to be $n_i/2$.

Example 7. Take the same polynomial as in 'Quick start guide':

$$\wp = (x - \hbar) \cdot (x - j) \cdot (x - i) = x^3 + (-i - j - \hbar) \cdot x^2 + (-i + j - \hbar) \cdot x + -1$$

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
> p := (x-kk)*(x-jj)*(x-ii); p;
x^3 + (-ii - jj - kk)*x^2 + (-ii + jj - kk)*x + -1
```

In the introductory section we saw that i is a root of $\wp$:

```
> test, a := HasRoot(p); test;
true
> Evaluate(p, a);
0
```

In fact, it is the only root of $\wp$ and its isolated multiplicity equals 3:

```
> Roots(p);
[ <ii, 3, 0> ]
```

Example 8. It is possible that both multiplicities (isolated and spherical) of a root of a quaternionic polynomial are simultaneously nonzero. Consider the quaternion division algebra $\mathscr{A} = \left(\frac{-1,-3}{K}\right)$, where $K = \mathbb{Q}(\sqrt{-2})$:

```
> K<sqrt3> := QuadraticField( -2 );
> A<ii,jj,kk> := QuaternionAlgebra<K | -1, -3>;
```

Take a polynomial

$$\wp = x^4 - (3 - i + j + \hbar) \cdot x^3 + 2 \cdot (2 - i + 2j + \hbar) \cdot x^2 - 2 \cdot (1 - i + 3j + \hbar) \cdot x + 4j$$

```
> Ax<x> := PolynomialRing(A);
> p_ := [
>   4*jj, -2*(1 - ii + 3*jj + kk), 2*(2 - ii + 2*jj + kk),
>   -(3 - ii + jj + kk), 1
> ];
> p := Ax!p_; p;
x^4 + (-3 + ii - jj - kk)*x^3 + (4 - 2*ii + 4*jj + 2*kk)*x^2
    + (-2 + 2*ii - 6*jj - 2*kk)*x + 4*jj
```

Let us find the roots of $\wp$ in $\mathscr{A}$:

```
> roots := Roots(p); roots;
[ <1 + 1/2*ii + 1/2*kk, 1, 2>, <jj + kk, 1, 0> ]
```

Verify the correctness of the result:

```
> [ Evaluate(p, a[1]) : a in roots ];
[ 0, 0 ]
```

Now, $\mathit{i} + \mathit{k}$ is an isolated root of multiplicity 1, but $a := 1 + {}^{(\mathit{i}+\mathit{k})}/_2$ has nonzero both the isolated and spherical multiplicities. In particular this means that every quaternion congruent to a must be a root of $\wp$, too. Take for example $b := (1 + \mathit{j}) \cdot a \cdot (1 + \mathit{j})^{-1}$:

```
> a := roots[1][1];
> b := (1 + jj)*a*(1 + jj)^-1; b;
1 + 1/2*ii - 1/2*kk
> Evaluate(p, b);
0
```

3.5. **Interpolation.**

`BrayWhaplesPolynomial( L )`

Let $\mathscr{A}$ be a quaternion division algebra. Bray-Whaples theorem (see, e.g., [12, Theorem 16.13]) says that for every pairwise non-congruent quaternions $a_1, \ldots, a_n \in \mathscr{A}$ there is a unique monic polynomial $\wp$ of degree $\deg \wp = n$ that vanished precisely at these quaternions.

This function comes in two variants. If $\mathscr{L}$ is either a set or a sequence of pairwise non-congruent quaternions, this function returns the monic polynomial, whose existence is asserted by the Bray-Whaples theorem. On the other hand, if $\mathscr{L}$ is a set or a sequence of triples $(a_i, e_i, \varepsilon_i)$ for $i \leq k$ such that:

- $a_1, \ldots, a_k$ are pairwise non-congruent quaternions,
- $e_1, \ldots, e_k$ are non-negative integers,
- $\varepsilon_1, \ldots, \varepsilon_k$ are even non-negative integers,
- if a_i is a central element, then ε_i is zero,
- for every i either e_i or ε_i is nonzero,

then the function returns a polynomial $\wp$ that has roots at $a_1, \ldots, a_k$ with isolated multiplicities $e_1, \ldots, e_k$ and spherical multiplicities $\varepsilon_1, \ldots, \varepsilon_k$. (See the description of the function `Roots` for the notion of isolated and spherical multiplicities.)

Example 9. We will use the quaternion algebra $\mathscr{A} = \left(\frac{-1,-1}{\mathbb{Q}}\right)$:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
```

Build a list of three quaternions:

```
> L := [ 1 + ii, 2 - jj, ii - 3*kk ];
```

Let $\wp \in \mathscr{A}[x]$ be the Bray–Whaples polynomial:

```
> p := BrayWhaplesPolynomial( L ); p;
x^3 + (-3 - 41/55*ii + 2/55*jj + 27/11*kk)*x^2
    + (259/55 + 19/11*ii - 1/11*jj - 43/5*kk)*x
    + -136/55 - 134/55*ii + 208/55*jj + 472/55*kk
```

Verify that it vanishes on every element of $\mathscr{L}$:

```
> [ Evaluate(p, l) : l in L ];
[ 0, 0, 0 ]
```

We can also compute the roots of $\wp$ in $\mathscr{A}$ (see the previous section for the descriprion of the function `Roots`) and check that these are precisely the quaternions in the list $\mathscr{L}$:

```
> [ a[1] : a in Roots(p) ];
[ 2 - jj, 1 + ii, ii - 3*kk ]
```

Example 10. Let as take the same quaternions $1 + i$, $2 - j$ and $i - 3k$ as in the previous example. This time, however, we will assign them some multiplicities. We want $1 + i$ to be a single isolated root, so its isolated multiplicity is 1 and spherical multiplicity is 0. Next, let $2 - j$ have a spherical multiplicity 2 and no isolated multiplicity. Finally, let $i - 3k$ be a root with the isolated and spherical multiplicities equal 2:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
> L := [ <1 + ii,1,0>, <2 - jj,0,2>, <ii - 3*kk,2,2> ];
```

Construct a polynomial $\wp$ qith the prescribed roots and multiplicities:

```
> p := BrayWhaplesPolynomial( L ); p;
x^7 + (-5 + 3/5*ii - 6/5*jj + 6*kk)*x^6
    + (53/5 - 4*ii - 156/5*kk)*x^5
    + (-57/5 + 127/5*ii + 6/5*jj + 594/5*kk)*x^4
    + (-76 - 88*ii - 24*jj - 348*kk)*x^3
    + (436 + 244*ii + 132*jj + 588*kk)*x^2
    + (-820 - 480*ii - 240*jj - 360*kk)*x
    + 500 + 500*ii
```

Execute the command `Roots` to verify the correctness of the result:

```
> Roots(p);
[ <2 - ii, 0, 2>, <ii - 3*kk, 2, 2>, <1 + ii, 1, 0> ]
```

`Interpolation( A, B )`

Let $\mathscr{A} = (a_1, \ldots, a_n)$ and $\mathscr{B} = (b_1, \ldots, b_n)$ be two lists quaternions of the same length. Assume that quaternions $a_1, \ldots a_n$ are **pairwise non-congruent**. This function returns a quaternionic polynomial $\wp$ such that $\wp(a_j) = b_j$ for every $j \leq n$.

Example 11. We will use the same quaternion algebra $\mathscr{A} = \left(\frac{-1,-1}{\mathbb{Q}}\right)$ and the same list $\mathscr{L}$ as in the previous example:

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
> L := [ 1 + ii, 2 - jj, ii - 3*kk ];
```

Now, build another list, say

```
> M := [ jj, 2, -3*kk ];
```

Construct the interpolating polynomial $\wp$:

```
> p := Interpolation(L, M); p;
(7/550 + 31/550*ii + 123/550*jj + 19/550*kk)*x^2
    + (511/550 + 183/550*ii - 169/550*jj + 173/550*kk)*x
    + -133/275 - 354/275*ii + 254/275*jj - 48/275*kk
```

Verify that indeed every quaternion from the list $\mathscr{L}$ is mapped by $\wp$ to the corresponding quaternion from $\mathscr{M}$:

```
> [ Evaluate(p, l) : l in L ];
[ jj, 2, -3*kk ]
```

3.6. **Factorization.**

`Factorization( p )`

`Factorisation( p )`

`Nice` BoolElt *Default:* `true`

Let $\mathscr{A}$ be a division quaternion algebra over a number field K. Given a quaternionic polynomial $\wp$, this function returns a factorization of $\wp$ as a sequence of pairs, each consisting of an irreducible factor $\mathfrak{q}_i$ and a positive integer k_i being its multiplicity. Each irreducible factor is monic. As a second output value the function returns the leading coefficient $c = \operatorname{lc}\wp$ of $\wp$. Thus, if the output is $\mathscr{L} = \big((\mathfrak{q}_1, k_1), \ldots, (\mathfrak{q}_n, k_n)\big)$ and c, then

$$\wp = c \cdot \mathfrak{q}_1^{k_1} \cdots \mathfrak{q}_n^{k_n}.$$

A quaternionic polynomial may have more than one factorization (potentially infinitely many). If the parameter `Nice` is set to `true` (the default), `Factorization` tries (reasonably hard) to find a factorization of $\wp$ with not too big coefficients. This behavior can be disabled by setting the parameter to `false`. The optimization of coefficients requires some extra effort, hence in theory setting `Nice` to `false` may potentially speed up the computations. In practice, the increase of the size of coefficients may easily consume any gains here. For randomly generated products of polynomials the default (i.e. `Nice := true`) resulted not only in smaller output but was also almost always faster.

Warning: We are in a non-commutative ring. The order of multiplication matters! To obtain back the original polynomial, the leading coefficient must be the left-most factor and all the other factors must be multiplied precisely in the order they appear within the output list. It is possible that a factor appears more than once in the list (see the example below).

Example 12. We will use the quaternion algebra $\mathscr{A} = \left(\frac{-1,-3}{K}\right)$, where $K = \mathbb{Q}(\sqrt{3})$.

```
> K<sqrt3> := QuadraticField( 3 );
> A<ii,jj,kk> := QuaternionAlgebra<K | -1, -3>;
> Ax<x> := PolynomialRing(A);
```

Consider the following quaternionic polynomial:

$$\wp = i \cdot x^6 + (j - \hbar) \cdot x^5 + (2i - 3) \cdot x^4 + (2j - 2\hbar) \cdot x^3 + (i - 6) \cdot x^2 + (j - \hbar) \cdot x - 3$$

```
> p_ := [ -3, jj - kk, ii - 6, 2*jj - 2*kk, 2*ii - 3, jj - kk, ii ];
> p := Ax!p_; p;
ii*x^6 + (jj - kk)*x^5 + (-3 + 2*ii)*x^4 + (2*jj - 2*kk)*x^3
       + (-6 + ii)*x^2 + (jj - kk)*x + -3
```

We can find its factorization in $\mathscr{A}[x]$:

```
> fp, c := Factorization(p); fp;
[
    <x + ii, 2>,
    <x + -ii, 2>,
    <x + -jj, 1>,
    <x + -kk, 1>
]
> c;
ii
```

The above MAGMA's output says that $\wp = i \cdot (x+i)^2(x-i)^2(x-j)(x-\hbar)$. Let us verify the correctness of the result multiplying all the constructed factors:

```
> q := c;
> for pi_ki in fp do
>     q *:= pi_ki[1]^pi_ki[2];
> end for;
> q eq p;
true
```

Warning: In the above example we use an explicit loop to multiply the factors and retrieve the original polynomial. This is due to the fact that MAGMA version 2.27-7 an earlier has a bug in the list multiplication operator `&*`. Consequently, neither one of the following two lines works correctly:

```
q := c * (&* [ pi_ki[1]^pi_ki[2] : pi_ki in fp ]);
q := c * FactorisationToPolynomial(fp);
```

Warning: Over a non-commutative ring, polynomial factorization is not unique! In general a polynomial may have infinitely many different factorizations. Anyway, every two factorizations have the same number of irreducible factors (counted with multiplicities). `Factorization` returns just one possible factorization.

Example 13. In this example we demonstrate the phenomenon of the non-uniqueness of polynomial factorization over quaternion algebras.

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
```

Define the quaternionic polynomial $\wp$ as a product of two linear (hence clearly irreducible) polynomials:

```
> p := (x - ii)*(x + jj + 1); p;
x^2 + (1 - ii + jj)*x + -ii - kk
```

Find its factorization:

```
> fp1, c1 := Factorization(p); fp1;
[
    <x + -ii, 1>,
    <x + 1 + jj, 1>
]
```

Now, we will find another factorization setting the parameter `Nice` to `false`:

```
> fp2, c2 := Factorization(p : Nice := false); fp2;
[
    <x + 1 - 2/3*ii + 1/3*jj + 2/3*kk, 1>,
    <x + -1/3*ii + 2/3*jj - 2/3*kk, 1>
]
```

The two obtained factors evidently differ from each other. Nonetheless, their products equal $\wp$ in both cases.

```
> p eq fp1[1][1]*fp1[2][1];
true
> p eq fp2[1][1]*fp2[2][1];
true
```

Example 14. With this example we show that the same factor may appear in a polynomial factorization more than once. Take the number field $K := \mathbb{Q}(\theta)$, where θ is a root of the polynomial $X^3 + 3X - 1$, and let $\mathscr{A} := \left(\frac{-1,-1}{K}\right)$:

```
> QX<X> := PolynomialRing(Rationals());
> K<theta> := NumberField( X^3 + 3*X - 1 );
> A<ii,jj,kk> := QuaternionAlgebra<K | -1, -1>;
```

Consider a quaternionic polynomial $\wp = x^4 - (3i+\hbar)\cdot x^3 - (3+j)\cdot x^2 + (i-\hbar)\cdot x - j$:

```
> Ax<x> := PolynomialRing(A);
> p := x^4 - (3*ii + kk)*x^3 - (3 + jj)*x^2 + (ii - kk)*x - jj; p;
x^4 + (-3*ii - kk)*x^3 + (-3 - jj)*x^2 + (ii - kk)*x + -jj
```

Find its factorization:

```
> Factorization(p);
[
    <x + -ii, 2>,
    <x + -kk, 1>,
    <x + -ii, 1>
]
1
```

As we can see, the factor $(x - i)$ appears twice (with multiplicities 2 and 1, respectively), separated by the factor $(x - \hbar)$.

3.7. **Other functions.**

`Resultant(p, q)`

`alg` MonStgElt *Default:* `"auto"`

Given two quaternionic polynomials $\wp, \mathfrak{q} \in \mathscr{A}[x]$ this function returns their resultant. Following [7] the resultant of non-commutative polynomials is defined to be the Dieudonné determinant of their Sylvester matrix. The resultant is an element of the base field K of $\mathscr{A}$. More precisely it is an element of the semi-ring $N(\mathscr{A}) \subset K$,

which is isomorphic to the co-domain of the Dieudonné determinant (i.e., the semiring constracted by appending zero to the abelianization of the unit group of $\mathscr{A}$). The resultant vanishes if and only if the two polynomials have a non-trivial common right factor.

The optional parameter `alg` selects the algorithm used to compute the resultant. The following values are accepted: "companion" (uses companion matrix to compute the resultant), "Euclid" (uses Euclidean algorithm), "Sylvester" (uses Sylvester matrix, ie. computes the resultant directly by definition) and "auto" (this is the same as "companion").

Example 15. Let $\mathscr{A} = \left(\frac{-1,-1}{\mathbb{Q}}\right)$. Take two polynomials

$$\wp = \hbar x^4 + (2+j)\cdot x^3 - (3-i)\cdot x^2 + (1-2j)\cdot x + 3j$$
$$\mathscr{q} = x^4 - (i+j)\cdot x^2 + \hbar$$

```
> QQ := Rationals();
> A<ii,jj,kk> := QuaternionAlgebra<QQ | -1, -1>;
> Ax<x> := PolynomialRing(A);
> p := kk*x^4 + (2 + jj)*x^3 - (3 - ii)*x^2 + (1 - 2*jj)*x + 3*jj;
> q := x^4 - (ii + jj)*x^2 + kk;
```

Compute their resultant:

```
> Resultant(p, q);
0
```

It is null. Hence, the two polynomials must share a common factor. We may verify it computing their greatest common right divisor.

```
> GCRD(p, q);
x^2 + -jj
```

`Sqrt(a)`

`SquareRoot(a)`

Given a quaternion $a \in \mathscr{A}$, this function outputs its square root. If a is not a square, then its execution results in an error.

Warning: Unlike most other functions, `Sqrt` works just with quaternions not quaternionic polynomials.

References

[1] Bernard Beck. Sur les équations polynomiales dans les quaternions. *Enseign. Math. (2)*, 25(3-4):193–201 (1980), 1979.

[2] Vladimir Bolotnikov. Lagrange interpolation over division rings. *Comm. Algebra*, 48(9):4065–4084, 2020.

[3] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[4] Una Bray and George Whaples. Polynomials with coefficients from a division ring. *Canad. J. Math.*, 35(3):509–515, 1983.

[5] John Cannon, Wieb Bosma, Claus Fieker, and Allan Steel (eds.). *Handbook of Magma Functions*, 2.27-7 edition, 2022.

[6] John Cannon and Catherine Playoust. *First Steps in Magma*. School of Mathematics and Statistics, University of Sydney, 1996.

[7] Aleksandra Lj. Erić. The resultant of non-commutative polynomials. *Mat. Vesnik*, 60(1):3–8, 2008.

[8] Graziano Gentili and Daniele C. Struppa. On the multiplicity of zeroes of polynomials with quaternionic coefficients. *Milan J. Math.*, 76:15–25, 2008.

[9] Basil Gordon and Theodore Samuel Motzkin. On the zeros of polynomials over division rings. *Trans. Amer. Math. Soc.*, 116:218–226, 1965.
[10] Basil Gordon and Theodore Samuel Motzkin. Correction to the paper "On the zeros of polynomials over division rings". *Trans. Amer. Math. Soc.*, 122:547, 1966.
[11] Bahman Kalantari. Algorithms for quaternion polynomial root-finding. *J. Complexity*, 29(3-4):302–322, 2013.
[12] Tsit-Yuen Lam. *A first course in noncommutative rings*, volume 131 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 2001.
[13] Takis Sakkalis, Kwanghee Ko, and Galam Song. Roots of quaternion polynomials: theory and computation. *Theoret. Comput. Sci.*, 800:173–178, 2019.
[14] Marie-France Vignéras. *Arithmétique des algèbres de quaternions*, volume 800 of *Lecture Notes in Mathematics*. Springer, Berlin, 1980.
[15] John Voight. *Quaternion algebras*, volume 288 of *Graduate Texts in Mathematics*. Springer, Cham, [2021] ©2021.

4. MIT License

Copyright (c) 2023 Przemysław Koprowski

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

> The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.