Shape functions based on the generic reference elements

ANDREAS DEDNER

A.S.Dedner@warwick.ac.uk,
www2.warwick.ac.uk/fac/sci/maths/people/staff/andreasd_dedner
Mathematics Institute,

THE UNIVERSITY OF WARWICK

Co-worker:
Martin Nolte

Stuttgart, 7th of October 2010
2003: Asked to join DUNE

*It is not possible to implement an efficient generic grid interface*
(A. Dedner, writing up his phd thesis)

2004: Asked to join DUNE again

*Ok, I will give it a try*
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).
2003: Asked to join DUNE

*It is not possible to implement an efficient generic grid interface*
(A. Dedner, writing up his PhD thesis)

2004: Asked to join DUNE again

*Ok, I will give it a try*
(A. Dedner, finished his PhD thesis)

I missed some of the initial fun (introduction of `const Entities` in 2004) but added some fun of my own (`GenericReferenceElements` in 2008).
My road to DUNE

2003: Asked to join DUNE

*It is not possible to implement an efficient generic grid interface*  
(A. Dedner, writting up his phd thesis)

2004: Asked to join DUNE again

*Ok, I will give it a try*  
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of *const Entities* in 2004) but added some fun of my own (*GenericReferenceElements* in 2008).
2003: Asked to join DUNE

It is not possible to implement an efficient generic grid interface
(A. Dedner, writing up his phd thesis)

2004: Asked to join DUNE again

Ok, I will give it a try
(A. Dedner, finished his phd thesis)

I missed some of the initial fun (introduction of const Entities in 2004) but added some fun of my own (GenericReferenceElements in 2008).
**Evolution Equations**

**Given:** domain $\Omega$, ansatz space $V$, test space $W$, quadratic form $a: V \times W \rightarrow \mathbb{R}$ (linear in second term), data $u_0 \in V$

**Problem:** find $u: (0, \infty) \times \Omega$ with $u(t, \cdot) \in V$, $u(0, x) = u_0(x)$, and

$$\int_{\Omega} \partial_t u(t, \cdot) \varphi = a(u(t, \cdot), \varphi) \quad \forall \varphi \in V$$

**Example:** Non-linear heat equation ($\partial_t u - \nabla \cdot A(u) \nabla u = 0$)

$\Omega = [0, 1]^2$, $V = H^1(\Omega)$ and

$$a(u, \varphi) = -\int_{\Omega} A(u) \nabla u \cdot \nabla \varphi$$
**Discrete Formulation of Evolution Equations**

\[ u(0, x) = u_0(x), \quad \int_{\Omega} \partial_t u(t, \cdot) \varphi = a(u(t, \cdot), \varphi) \quad \forall \varphi \in V \]

**Construct:** tessellation \( \mathcal{G} \) of \( \Omega \), discrete function space \( V_\mathcal{G} \subset V \),

projection \( u_{\mathcal{G},0} = \Pi_\mathcal{G}[u_0] \in V_\mathcal{G} \)

**Problem (semi-discrete):** find \( u_{\mathcal{G}} : (0, \infty) \rightarrow V_\mathcal{G} \) with \( u_{\mathcal{G}}(0) \equiv u_{\mathcal{G},0} \), and

\[ \sum_{E \in \mathcal{G}} \int_E \partial_t u_{\mathcal{G}}(t) \varphi_{\mathcal{G}} = a(u_{\mathcal{G}}(t), \varphi_{\mathcal{G}}) \quad \forall \varphi_{\mathcal{G}} \in V_\mathcal{G} \]

Using a basis \( \mathcal{B} \) of \( V_\mathcal{G} \) we can write \( u(t) = \sum_{\psi \in \mathcal{B}} u_{\psi}(t) \psi \) where \( u_{\psi}(t) : [0, \infty) \rightarrow \mathbb{R} \) are the degrees of freedom (DoF):

\[ \sum_{\psi \in \mathcal{B}} \frac{d}{dt} u_{\psi}(t) \sum_{E \in \mathcal{G}} \int_E \psi \varphi = a(u_{\mathcal{G}}(t), \varphi) \quad \forall \varphi \in \mathcal{B} \]

**Problem (fully discrete):** The semi-discrete formulation leads to a system of ordinary differential equations for the DoFs \( (u_{\psi})_{\psi \in \mathcal{B}} \). Use of an ODE solver to obtain fully discrete scheme (method of lines)
Discrete Formulation of Evolution Equations

Requires:

1. construct tessilation $\mathcal{G}$ of $\Omega$
2. construct basis $\mathcal{B}$ of $V_\mathcal{G}$
3. implement of quadratic forms $a(\cdot, \cdot)$ with implementation of efficient operations $\ +, \circ$ etc.
4. implement ODE solver (e.g. explicit, implicit, semi-implicit)

Need to:

1. evaluate basis function and derivatives
2. efficient evaluation in given set of points (quadrature)
3. interpolation and projection into discrete function space
4. associate basis functions with subentities
Given: grid $\mathcal{G}$ with entities $E$ and a small set of reference elements $\mathcal{R}$

Assumption: for each entity $E$ there is a reference element $\hat{E} \in \mathcal{R}$
and a bijective smooth mapping $F_E : \hat{E} \rightarrow E$
Localized Discrete Function Space

Shape functions: for each $\hat{R} \in \mathcal{R}$ there is a shape function set $\hat{B}_{\hat{R}}$

**Task 1:** construct basis function set $\mathcal{B}$ of $V_G$ with for all
\[ \psi \in \mathcal{B} \text{ and } E \in \mathcal{G}: \psi|_E = \hat{\psi} \circ F^{-1}_E \text{ for some } \psi \in \hat{B}_{\hat{R}} \]

**Task 2:** construct a mapper $\mu_E: \hat{B}_{\hat{E}} \rightarrow \mathcal{B}$ so that
\[ \mu_E(\hat{\psi})|_E = \hat{\psi} \circ F^{-1}_E \]

Then given $u_G = \sum_{\psi \in \mathcal{B}} u_{\psi} \psi \in V_G$ we have for $x \in E$
\[ u_E(x) = \sum_{\hat{\psi} \in \hat{B}_{\hat{E}}} u_{\mu_E(\hat{\psi})}(\hat{\psi} \circ F^{-1}_E)(x) \]

or using local coordinates $\hat{x} \in \hat{E}$:
\[ \hat{u}_E(\hat{x}) = \sum_{\hat{\psi} \in \hat{B}_{\hat{E}}} u_{\mu_E(\hat{\psi})}\hat{\psi}(\hat{x}) \]

**Advantage:**
simple construction of space $V_G$ and speedup through caching of $\hat{\psi}$. 
Given set of reference elements $\mathcal{R}^d$ with $R \subset \mathbb{R}^d, R \in \mathcal{R}^d$ we define

$$\mathcal{R}^{d+1} = \{ R|, R^\circ : R \in \mathcal{R}^{d+1} \}$$

where for $R \in \mathcal{R}^d$:

$$R| = \{(x, z) : z \in [0, 1], x \in R\}$$

$$R^\circ = \{(x(1 - z), z) : z \in [0, 1], x \in R\}$$

For $d = 0$ we set

$$\mathcal{R}^0 = \{P\} \text{ with } P = \{0\} \in \mathbb{R}^0.$$

**Note:** $R^\circ$ is the Duffy transform of $R$.

**Note:** $R|| = R^\circ|, R^\circ\circ = R|^\circ, \ldots$.

**Advantage:**

Use of *template meta program* to construct numbering of subentities, mappings, and much more.
Template Meta Programs

**Static identification**

```cpp
struct Point;
template <class Base> struct Prism;
template <class Base> struct Pyramid;
```

**Dynamic identification (Topologyid)**

Identify $R$ by pair of positive numbers $(\text{topologyId}(R), \text{dimension}(R))$:

- $\text{topologyId}(R^\circ) = \text{topologyId}(R)$,
- $\text{topologyId}(R|) = 2^{\text{dim}-1} + \text{topologyId}()$.

**IfTopology**

Convert dynamic topologyId($R$) into static type

```cpp
template <class Topology> struct Operation {
    static void apply(...) {
    }
};

IfTopology<Operation, dimension>::apply(topologyId, ...);
```

Using a *dynamic* topologyId, calls $\text{Operation<Topology>::apply(...);}$ with the correct *static* topology class.
Storage

template <class Traits> struct TopologyFactory {
    static Traits::Object*
        create(unsigned int topologyId, const Traits::Key &key) {
            Object *object;
            IfTopology<Factory, Traits::dimension>::
                apply(topologyId, key, object);
            return object;
        }
};

calls Factory<Topolgy> to generate an instance of Object for a given
dynamic topologyId, the construction of which depends staticly on the
Topology.

Example: quadrature point sets

Wrapper for singleton storage

template <class Factory>
struct TopologySingletonFactory;
template <class Topology>
struct Factory;

template <>
struct Factory <Point> {
    Object *create(const Key &key) { ... };
};

template <class Base>
struct Factory <Pyramid <Base>> {
    Object *create(const Key &key) { ... };
};

template <class Base>
struct Factory <Prism <Base>> {
    Object *create(const Key &key) { ... };
};
Local Interfaces (static)

Basis

unsigned int size () const;
unsigned int order () const;
inline void evaluateFunction (  
    const typename Traits::DomainType& in,
    std::vector<typename Traits::RangeType>& out ) const;
inline void evaluateJacobian (  
    const typename Traits::DomainType& in,
    std::vector<typename Traits::JacobianType>& out ) const;
inline void evaluate (  
    const typename Dune::template array<int, Traits::diffOrder>& direct,
    const typename Traits::DomainType& in,
    std::vector<typename Traits::RangeType>& out ) const;

Interpolation

template <class F, class C>  
void interpolate ( const F& f, std::vector<C>& out ) const;
Local Interfaces

Key

```
std::size_t size() const;
const LocalKey& localKey(std::size_t i) const;

struct LocalKey {
    unsigned int subEntity() const;
    unsigned int codim() const;
    unsigned int index() const;
};
```

Note

All interface exist in as virtual interface for use with different reference elements...
First approach: static interface -> construct virtual interface using wrapper (e.g. with TopologyFactory)
Second approach: use virtual interface directly
Construction of General Shape Function Set

1. Generic implementation of monomial basis of order $p$:
   Given $M^p_R$ for all $R \in \mathcal{R}^d$, define $M^p_R\mid$, $M^p_R\circ$.
   
   **Note:** $M^p_R$ is set needed for Lagrange interpolation on $R$ of order $p$.

2. Use $M^p_R$ to define a set of (vectorial) polynomial functions $\tilde{B} = \{\tilde{b}_j\}$

3. Construct matrix $A$ and define final set: $B = A \tilde{B}$

**Example:**
Lagrange space is defined through Lagrange interpolation $\lambda^p_i(u) = u(x^p_i)$
where $x^p$ are the Lagrange points of order $p$.
Use $\tilde{B} = M^p_R$ and define $B = (\lambda^p_i(\tilde{b}_j))_{ij}$ then with $A = B^{-T}$, we have
$\lambda^p_i(b_j) = \delta_{ij}$ for $B^p_R = \{b_j\}$

*template meta program* over reference elements to construct lagrange points
**Ansatz:** we assume each base function is a polynomial in $d$ variables.

**Example on simplex topology** $S^0 = P$, $S^{d+1} = (S^d)^\circ$

We construct $\Psi^d_k$ of all monomials in $d$ variables of exactly order $k$:

<table>
<thead>
<tr>
<th>dim. $k$</th>
<th>dim. monomials</th>
<th>recursion relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\Psi^0_0 = 1$</td>
<td>$\Psi^0_0 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi^1_0 = 1$</td>
<td>$\Psi^0_0 = 1$</td>
</tr>
<tr>
<td>1</td>
<td>$\Psi^1_1 = x$</td>
<td>$\Psi^0_1 = 0$, $x\Psi^0_0 = x$</td>
</tr>
<tr>
<td></td>
<td>$\Psi^1_2 = x^2$</td>
<td>$\Psi^0_2 = 0$, $x\Psi^0_1 = 0$, $x(x\Psi^0_0) = x^2$</td>
</tr>
<tr>
<td>2</td>
<td>$\Psi^2_0 = 1$</td>
<td>$\Psi^1_0 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi^2_1 = {x, y}$</td>
<td>$\Psi^1_1 = x$, $= y$</td>
</tr>
<tr>
<td></td>
<td>$\Psi^2_2 = {x^2, xy, y^2}$</td>
<td>$\Psi^1_2 = x^2$, $y\Psi^1_1 = xy$, $y$, $= y^2$</td>
</tr>
</tbody>
</table>

$\Psi^{d+1}_0 = \{\ldots\}$

$\Psi^{d+1}_1 = \{\ldots\}$

$\Psi^{d+1}_2 = \{\ldots\}$

Slight modification required for general reference element $R^\circ$.

**Note:** recursion also works for arbitrary derivatives
Ansatz: we assume each base function is a polynomial in $d$ variables.

Example on simplex topology $S^0 = P$, $S^{d+1} = (S^d) \circ$

We construct $\Psi^d_k$ of all monomials in $d$ variables of exactly order $k$:

<table>
<thead>
<tr>
<th>dim.</th>
<th>monomials</th>
<th>recursion relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\Psi_0^0 = 1$</td>
<td>$\Psi_0^0 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^1 = 1$</td>
<td>$\Psi_1^1 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^0 = x$</td>
<td>$\Psi_1^0 = 0$ $x\Psi_0^0 = x$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_2^1 = x^2$</td>
<td>$\Psi_2^0 = 0$ $x\Psi_1^0 = 0$ $x(x\Psi_0^0) = x^2$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_2^2 = 1$</td>
<td>$\Psi_0^1 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^2 = {x, y}$</td>
<td>$\Psi_1^1 = x$ $y\Psi_0^1 = y$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_2^2 = {x^2, xy, y^2}$</td>
<td>$\Psi_1^2 = x^2$ $y\Psi_1^1 = xy$ $y(y\Psi_0^1) = y^2$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_{d+1}^d = {\ldots}$</td>
<td>$\Psi_0^d$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_{d+1}^d = {\ldots}$</td>
<td>$\Psi_1^d \ z \Psi_0^d$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_{d+1}^d = {\ldots}$</td>
<td>$\Psi_2^d \ z \Psi_1^d \ z (z \Psi_0^d)$</td>
</tr>
</tbody>
</table>

Slight modification required for general reference element $R^\circ$.

Note: recursion also works for arbitrary derivatives
**Generic Monomials**

**Ansatz:** we assume each base function is a polynomial in $d$ variables.

**Example on simplex topology** $S^0 = P, S^{d+1} = (S^d)^\circ$

We construct $\Psi^d_k$ of all monomials in $d$ variables of exactly order $k$:

<table>
<thead>
<tr>
<th>dim. monomials</th>
<th>recursion relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$\Psi_0^0 = 1$</td>
</tr>
<tr>
<td>$\Psi_1^0 = 1$</td>
<td>$\Psi_0^0 = 1$</td>
</tr>
<tr>
<td>$\Psi_1^1 = x$</td>
<td>$\Psi_1^0 = \emptyset$ $x \Psi_0^0 = x$ $x(x \Psi_0^0) = x^2$</td>
</tr>
<tr>
<td>$\Psi_2^1 = x^2$</td>
<td>$\Psi_2^0 = \emptyset$ $x \Psi_1^0 = \emptyset$ $x(x \Psi_0^0) = x^2$</td>
</tr>
<tr>
<td>$\Psi_0^2 = 1$</td>
<td>$\Psi_0^1 = x$ $y \Psi_0^0 = y$ $y(y \Psi_0^1) = y^2$</td>
</tr>
<tr>
<td>$\Psi_1^2 = {x, y}$</td>
<td>$\Psi_1^2 = x^2$ $y \Psi_1^1 = xy$ $y(y \Psi_0^1) = y^2$</td>
</tr>
<tr>
<td>$\Psi_2^2 = {x^2, xy, y^2}$</td>
<td>$\Psi_2^2 = {\ldots}$ $z \Psi_0^d$ $z(z \Psi_0^d)$</td>
</tr>
<tr>
<td>$\Psi_0^{d+1} = {\ldots}$</td>
<td>$\Psi_0^{d+1} = {\ldots}$ $z \Psi_0^d$</td>
</tr>
<tr>
<td>$\Psi_1^{d+1} = {\ldots}$</td>
<td>$\Psi_1^{d+1} = {\ldots}$ $z \Psi_0^d$</td>
</tr>
<tr>
<td>$\Psi_2^{d+1} = {\ldots}$</td>
<td>$\Psi_2^{d+1} = {\ldots}$ $z \Psi_0^d$ $z(z \Psi_0^d)$</td>
</tr>
</tbody>
</table>

Slight modification required for general reference element $R^\circ$.

**Note:** recursion also works for arbitrary derivatives
### Generic Monomials

#### Example on cube topology $Q^0 = P$, $Q^{d+1} = (Q^d)_1$

We construct $\Psi_k^d$ of all bi-monomials in $d$ variables of exactly order $k$:

<table>
<thead>
<tr>
<th>dim.</th>
<th>bi-monomials</th>
<th>recursion relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\Psi_0^0 = 1$</td>
<td>$\Psi_0^0 = 1$</td>
</tr>
<tr>
<td>1</td>
<td>$\Psi_0^1 = 1$</td>
<td>$\Psi_0^0 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^1 = x$</td>
<td>$\Psi_0^0 = \emptyset$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x\Psi_0^0 = \emptyset$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$x\Psi_0^0 = x$</td>
</tr>
<tr>
<td>2</td>
<td>$\Psi_0^2 = 1$</td>
<td>$\Psi_0^1 = 1$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^2 = {x, xy, y}$</td>
<td>$\Psi_0^1 = x$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$y\Psi_0^1 = yx$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$y\Psi_0^1 = y$</td>
</tr>
<tr>
<td>d+1</td>
<td>$\Psi_0^{d+1} = {\ldots}$</td>
<td>$\Psi_0^d$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_1^{d+1} = {\ldots}$</td>
<td>$\Psi_1^d$</td>
</tr>
<tr>
<td></td>
<td>$\Psi_2^{d+1} = {\ldots}$</td>
<td>$\Psi_2^d$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$z\Psi_1^d$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$z\Psi_1^d$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$z(z\Psi_2^d)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$z(z\Psi_1^d)$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$z(z\Psi_0^d)$</td>
</tr>
</tbody>
</table>

Recursion correct for any reference element $R^1$.

Use *template meta program* over the definition of the generic reference elements.
Example Implementation

template <class Topology> class Monomials;

template <class BaseTopology>
class Monomials < Pyramid < BaseTopology >> {
    static const int d = BaseTopology::dimension;
    static void evaluate(Domain x, Range m) {
        Monomials< BaseTopology >::evaluate(x, m);
        // fill right column, multiplying by z = x[d]
    }
};

template <class BaseTopology>
class Monomials < Prism < BaseTopology >> {
    static const int d = BaseTopology::dimension;
    static void evaluate(Domain x, Range m) {
        Monomials< BaseTopology >::evaluate(x, m);
        // fill right column, multiplying by z = x[d]
    }
};

template <>
class Monomials < Point > {
    static void evaluate(Domain x, Range m) {
        m[0] = 1.;
    }
};

Similar for quadratures, lagrange point sets...
Construction of General Basis Function Set

**Setting 1:**
Given functionals $\Lambda = (\lambda_i)_i$ and polynomial function set $\tilde{B} = (\tilde{b}_j)_j$:

1. define matrix $B = (\lambda_i(\tilde{b}_j))_{ij}$
2. construct $A = B^{-T}$ (using AlgLib with multiprecision arithmetic)
3. basis $\mathcal{B} = B^{-T}\tilde{B}$ satisfies $\Lambda(\mathcal{B}) = I$

**Example:** Raviart-Thomas space (arbitrary dimension and order)
vector valued functions on simplex reference element $R$

$$\tilde{B} = [\Psi^d]^d + x\Psi^d_p$$

$$\lambda_i(u) = \int_{\partial R} u \cdot n \varphi_i, \quad \lambda_{N+i}(u) = \int_R u \psi_i$$

where $\varphi_1, \ldots, \varphi_N$ is basis of $P_k(\partial R)$ and $\psi_1, \ldots, \psi_L$ is basis of $P_{k-1}(R)$

**Setting 2:**
given a bilinear form $a$ (e.g. $a(u, v) = \int_R uv$) construct orthonormal basis starting from $\tilde{B}$. Requires QR factorization of $(a(\tilde{b}_i, \tilde{b}_j))_{ij}$. 
Usage of high precision field type

1. Construction phase
Evaluate prebasis $\tilde{B}$ for $B = (\lambda_i(\tilde{b}_j))_{ij}$, compute $B^{-T}$ (QR factorization).

2. Evaluation phase
Evaluation of prebasis $\tilde{B}$ to compute basis functions $B = B^{-T}\tilde{B}$.

Note: For any basis the main step during evaluation is always the same matrix-vector multiplication (even for derivatives using derivatives of $\tilde{B}$). Setting up the matrix is only done once.

Usage of different field types:
We use high precision floating point arithmetics (alglib based on mpfr, gmp).
ComputeField: used to setup matrix and during inversion/QR.
StorageField: used for storing the matrix $B^{-T}$.
Evaluation is possible in any field (high for caching, double for on-the-fly...)
Conversion between different field types is transparent for the user using the field_cast mechanism:

```c
field_cast( in, out ); out = field_cast< OutField > ( in );
```
Further examples

- spectral Lagrange elements
- Raviart-Thomas using Lagrange points
  (still need implementation on general Prism type topologies)
- Nedelec edge elements
  (need to be implemented)
To Do (general)

dune module dependency

Now

dune-common

ReferenceElement
Geometry

Shape functions for geometries

FieldVector ...

dune-grid

dune-localfunction

Now

dune-common

ReferenceElement
Geometry

Shape functions for geometries

FieldVector ...

dune-grid

dune-localfunction

dune-geometry

global interface

1. Take reference mapping into account
2. Piola transform, orientation of normals...
3. Global orientation of subentities (twists)
replace monomials with more stable basis

code generation

Many possibilities: write Maple code and Maple generates optimized code.

Example:

\[
\varphi_1(a, b) = (-0.2424871130596428210938424878108221313725E1 a + 0.4156921938165305504465871219614093680664E1 a^2 + 0.353338364744059678795990536671979628557E2 ab - 0.5819690713431427706225219707459731152920E2 a^2 b - 0.9456997409326070022659857024622063123519E2 ab^2 + 0.116398142686285541250443941491946230584E3 a^2 b^2 + 0.5819690713431427706225219707459731152920E2 ab^3, 0.1732050807568877293527446341505872366945E1 - 0.3464101615137754587054892683011744733891E1 a - 0.2875204340564336307255560926899748129126E2 b + 0.5403998519614897155805632585498321784868E2 ab + 0.1070407399077566167399961839050629122771E3 b^2 - 0.1600414946193642619219360419551426067056E3 ab^2 - 0.13821765444399640802349021805216861488824E3 b^3 + 0.116398142686285541250443941491946230584E3 ab^3 + 0.5819690713431427706252219707459731152920E2 b^4)
\]

\[
\varphi_2(a, b) = (0.491934955049953732100182071208807717973E1 a - 0.2146625258399798108552806721982025186026E2 a^2 - 0.4561578674099570980674714284211803520299E2 ab + 0.18782971010998234498370581734272037769E3 a^2 b + 0.6574039853849381707442970586069952132183E2 ab^2 - 0.1502637680879858675986964705387417630217E3 a^3 b - 0.1502637680879858675986964705387417630217E3 a^2 b^2 - 0.2504396134799764459978274508979029383691E2 ab^3,...)
\]