The Linnea Language

February 16, 2021

1 Introduction

Linnea is a prototype of a compiler that translates the mathematical description of a linear algebra problem to an efficient sequence of calls to BLAS and LAPACK kernels. This document describes the language for the input to Linnea.

2 The Linnea Language

The Linnea language consists of three parts: The definition of dimensions, the definition of operands, and linear algebra expressions.

2.1 Definition of Dimensions

Dimensions are defined by assigning a positive integer to a variable:

```
n = 1000
m = 2000
```

2.2 Definition of Operands

As operands, matrices, vectors, and scalars can be used. The definition of an operand consists of up to four parts: The type of the operand, the name, the dimensions, and the properties. As an example, the definition of a matrix `M` with dimensions `m` by `n` and the property `FullRank` is written as:

```
Matrix M(m, n) <FullRank>
```

Notice that for specifying the dimensions, variables as described in Sec. 2.1 have to be used. There are three different types of matrices: General matrices (`Matrix`), the identity matrix (`IdentityMatrix`), and the zero matrix (`ZeroMatrix`). General matrices can be annotated with an arbitrary number of properties (properties are discussed in detail in Sec. 2.4). Identity and zero matrices cannot be annotated with additional properties.

There are two types of vectors: Row vectors (`RowVector`) and column vectors (`ColumnVector`). A column vector `v` with length `n` is defined as:

```
ColumnVector v(n) <>
```

In contrast to matrices, vectors only require one dimensions; their lengths. Mathematically, a row vector with length `n` can be seen as a `1` by `n` matrix, a column vector as a `n` by `1` matrix.

A scalar `alpha` with the property `Positive` is defined as:

```
Scalar alpha <Positive>
```

2.3 Expressions

The actual mathematical problem is described as a sequence of assignments, where the left-hand side of each assignment is a single operand, and the right-hand side is an expression built from addition (`+`), multiplication (`*`), subtraction (`-`), transposition (`trans(A)`), and inversion (`inv(A)`). In addition, constant scalars such as `-1` or `0.5` can be used. Three examples are shown below:

1. `X = alpha*trans(A)*B + 2*C`
2. `b = inv(trans(X)*X)*trans(X)*y`

3.
   ```
   H_dag = trans(H)*inv(H*trans(H))
   y_k   = H_dag*y + (I_n - H_dag*H)*x
   ```

Operands are considered as unique, mathematical objects. For this reason, the value of an operand cannot change, and it is not possible to assign to an operand more than once. Which operands are input and which ones are output is determined from the sequence of assignments. All operands that appear on the left-hand side of an assignment are considered output, all other operands are input. As an example, in

```
X = A*B
Y = C+D
```

`A`, `B`, `C`, and `D` are input, `X`, and `Y` are output. Once a value has been assigned to an output operand, the operand can be used on the right-hand side in subsequent assignments:

```
X = A*B
Y = C+X
```

2.3.1 Validity of Expressions

For an expression to be valid, Linnea applies the same rules that are used in linear algebra:

1. The dimensions of the left- and right-hand side of an assignment have to be the same.
2. The dimensions of all operands in a sum have to be the same.
3. In a product `A*B`, where `A` and `B` are matrices or vectors, the number of columns of `A` has to be the same as the number of rows of `B`. Inner products are an exception. For instance, let v be a column vector and `A` be a matrix. The product `trans(v)*v*A` is valid because `trans(v)*v` forms an inner product and evaluates to a scalar. Notice that it is not necessary to use parenthesis around inner products; they are identified automatically.
4. Singular matrices or expressions cannot be inverted. An expression is considered singular if it is not possible to infer that it has the property `Non-singular` (see Sec. 2.4).

2.3.2 Influence of Different Representations

For most expressions, multiple mathematically equivalent representation exist. As a trivial example, since addition is commutative, the sum of `A` and `B` can be written as `A+B` and `B+A`. As another example, both in sums and products, different parenthesizations are possible: A product of three matrices can be written as `A*B*C`, `A*(B*C)`, and `(A*B)*C`. In most cases, such different representations of the input expressions do not affect the generated code. Instead, many different representations are explored by Linnea, and the one is used that leads to the best algorithm. Internally, input expressions are in a first step converted to a canonical representation, and not all possible representations are explored. As a result, it is possible that Linnea does not make use of a particularly favorable representation of the input provided by the user. However, those cases should be rare and only affect especially large and complicated expressions.

Both the generation time and the generated code is affected by breaking down expressions into multiple assignments, and conversely plugging in intermediate operands. The reasons is that operands on the left-hand side of an assignment are always explicitly computed and returned as output. As an example, breaking down the assignment

```
X = A*B*C
```

into two assignments

```
Y = A*B
X = Y*C
```

Property	Definition
`Diagonal`	$a_{ij} = 0$ if $i \neq j$
`LowerTriangular`	$a_{ij} = 0$ if $i < j$
`UpperTriangular`	$a_{ij} = 0$ if $i > j$
`UnitDiagonal`	$a_{ij} = 1$ if $i = j$ (requires `UpperTriangular` or `LowerTriangular`)
`Symmetric`	$m = n$ and $a_{ij} = a_{ji}$
`SPD`	$m = n$ and $x^T A x > 0$ for all $x \in \mathbb{R}^n$
`SPSD`	$m = n$ and $x^T A x \geqslant 0$ for all $x \in \mathbb{R}^n$
`Orthogonal`	$m = n$ and $A^T A = A A^T = I$
`OrthogonalColumns`	$m \geqslant n$ and $A^T A = I$
`OrthogonalRows`	$n \geqslant m$ and $A A^T = I$
`Permutation`	$m = n$ and there is exactly one 1 in each row and column, all other entries are 0
`Positive`	$\alpha > 0$ (only for scalars)
`FullRank`	A has full rank
`Non-singular`	$m = n$ and A has full rank

Table 1: Mathematical definitions of properties supported by Linnea. Let $A \in \mathbb{R}^{m \times n}$ be a real matrix. The elements of this matrix are denoted as a_{ij} with $i \in \{0, \dots, m-1\}$ and $j \in \{0, \dots, n-1\}$. For scalars, α is used.

forces Linnea to explicitly compute `Y`, which is equivalent to using the parenthesization `(A*B)*C`. If the optimal parenthesization is `A*(B*C)`, then the intermediate operand `Y` prevents Linnea from finding the optimal solution. This should be considered when using intermediate operands that are only used to simplify the input expressions and do not have to be computed explicitly.

In general, plugging in such intermediate operands may allow Linnea to find better solutions. At the same time, it increases the size of the search space and may increase the generation time. On the other hand, breaking down a large and complex expression into multiple smaller assignments can be used to decrease the generation time, at the cost of potentially suboptimal solutions.

2.4 Properties

Operands can be annotated with the properties shown in Tab. 1. It is possible for operands to have more than one property, as long as their mathematical definitions do not contradict one another. For instance, a matrix can be diagonal and SPD, which implies that all diagonal elements are positive.

It is not necessary to annotate operands with redundant properties. As an example, it is not necessary to also use the property `Symmetric` for a matrix that is already has the property `SPD`; such properties are inferred automatically. This includes properties that follow from the combination of a property with a specific size. For instance, a matrix that is square and has the property `Diagonal` is automatically identified as symmetric.

If intermediate operands are used in subsequent assignments, their properties are inferred from the right-hand side of their assignment. As an example, consider the following assignments:

```
X = trans(A)*A
Y = X*B
```

It is not necessary to specify that `X` is symmetric. Instead, this property is inferred from `trans(A)*A`. It is not possible to specify properties of output operands.

It is not checked whether the actual content of the input operands has the same properties that the operands were annotated with; it is up to the user to ensure that the specified properties hold. If the input operands do not have the specified properties, most likely the generated code produces incorrect results.

3 Grammar

The grammar of the Linnea language is shown in Fig. 1.

```
      model  =  {size_decl}, {op_decl}, {assignment};
  size_decl  =  id, "=", int;
    op_decl  =  "Matrix", id, dim_matrix, properties
             |  "RowVector", id, dim_vector, properties
             |  "ColumnVector", id, dim_vector, properties
             |  "Scalar", id, properties
             |  "IdentityMatrix", id, dim_matrix
             |  "ZeroMatrix", id, dim_matrix;
 dim_vector  =  "(", id, ")";
 dim_matrix  =  "(", id, ",", id, ")";
 properties  =  "<", [property, {",", property}], ">";
 assignment  =  id, "=", expr;
       expr  =  term, {( "+" | "-" ), term};
       term  =  factor, {"*", factor};
     factor  =  "(", expr, ")"
             |  "trans(", expr, ")"
             |  "inv(", expr, ")"
             |  "-", factor
             |  number
             |  id;
         id  =  ?"[a-zA-Z_][a-zA-Z0-9_]*"?;
     number  =  ?"[0-9]+(\.[0-9]+)?([Ee][+-]?[0-9]+)?"?;
        int  =  ?"[0-9]+"?;
```

Figure 1: Grammar of the Linnea language in extended Backus–Naur form. The nonterminal *property* matches the properties defined in Tab. 1. The strings enclosed in ?"… "? are regular expressions in the syntax of the Python module `re`.