

# LABMATE: Supporting Types for MATLAB

Conor McBride<sup>1</sup>, Georgi Nakov<sup>1</sup>, Fredrik Nordvall Forsberg<sup>1</sup>,  
André Videla<sup>1</sup>, Alistair Forbes<sup>2</sup>, Keith Lines<sup>2</sup>

<sup>1</sup>University of Strathclyde, UK

<sup>2</sup>National Physical Laboratory, UK

August 26, 2024

# The Problem

- ▶ Much science and engineering software is written in **MATLAB**.
  - ▶ May contain **errors** and **bugs**, as with any software.

# The Problem

- ▶ Much science and engineering software is written in **MATLAB**.
  - ▶ May contain **errors** and **bugs**, as with any software.
- ▶ Developers often leave comments about how their **data** should be **interpreted**, e.g., units of measure for quantities.

# The Problem

- ▶ Much science and engineering software is written in `MATLAB`.
  - ▶ May contain `errors` and `bugs`, as with any software.
- ▶ Developers often leave comments about how their `data` should be `interpreted`, e.g., units of measure for quantities.
- ▶ However `MATLAB` is `oblivious` to these high-level comments, and instead performs `low-level checks` during execution.

# Our Plan

*Can we do better?*

- ▶ Make these developer comments formal.

# Our Plan

*Can we do better?*

- ▶ Make these developer comments formal.
- ▶ ...and create a tool to make use of them — LABMATE.
  - ▶ Keep existing MATLAB code and toolchains; no need to switch to a new language.

# Our Plan

*Can we do better?*

- ▶ Make these developer comments formal.
- ▶ ...and create a tool to make use of them — LABMATE.
  - ▶ Keep existing MATLAB code and toolchains; no need to switch to a new language.
- ▶ Distill the essence of the developer comments in LABMATE's expressive type system.
  - ▶ A set of logical rules that assign domains of admissible values to program expressions.

# Our Plan

*Can we do better?*

- ▶ Make these developer comments formal.
- ▶ ...and create a tool to make use of them — LABMATE.
  - ▶ Keep existing MATLAB code and toolchains; no need to switch to a new language.
- ▶ Distill the essence of the developer comments in LABMATE's expressive type system.
  - ▶ A set of logical rules that assign domains of admissible values to program expressions.
- ▶ LABMATE is meant to be used while writing the code to get instant feedback and guidance — do not delay until execution.



## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

## How does LABMATE Work?

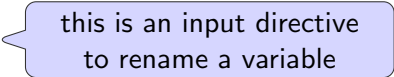
- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```



this is an input directive  
to rename a variable

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

- ▶ Run LABMATE to get:

```
%< LabMate 0.2.0.1  
%< renamed n x  
x = 5;  
display(x);
```

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

- ▶ Run LABMATE to get:

```
%< LabMate 0.2.0.1  
%< renamed n x  
x = 5;  
display(x);
```

## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

- ▶ Run LABMATE to get:

```
%< LabMate 0.2.0.1  
%< renamed n x  
x = 5;  
display(x);
```

LabMate response to  
the input directive



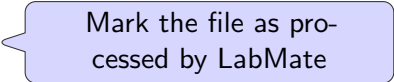
## How does LABMATE Work?

- ▶ LABMATE is a **program transducer**: reads MATLAB code with formal comments, and outputs a modified version of the input.
- ▶ These formal comments are **directives** — they start with **%<**.
- ▶ Input the program:

```
%> rename n x  
n = 5;  
display(n);
```

- ▶ Run LABMATE to get:

```
%< LabMate 0.2.0.1  
%< renamed n x  
x = 5;  
display(x);
```



Mark the file as processed by LabMate

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
A = [ 3 4 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      5 6 7 8 ]
C = A * B
```

type annotations  
at declaration of A

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

type annotations  
at declaration of B

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

```
%< C :: [Matrix 1 4 int]
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

```
%< C :: [Matrix 1 4 int]
```



# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

query for the type of C

```
%< C :: [Matrix 1 4 int]
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
A = [ 3 4 ]
%> B :: [ 2 x 4 ] int
B = [ 1 1 1 1
      5 6 7 8 ]
C = A * B
```

- ▶ We can ask for type information. LabMate can infer the dimensions

```
%> typeof C
%< C :: [Matrix 1 4 int]
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

```
%< C :: [Matrix 1 4 int]
```

- ▶ ...and can easily spot incompatible sizing:

```
D = B * A
```

```
%> typeof D
```

```
%< The expression D is quite a puzzle
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

```
%< C :: [Matrix 1 4 int]
```

- ▶ ...and can easily spot incompatible sizing:

```
D = B * A
```

```
%> typeof D
```

```
%< The expression D is quite a puzzle
```

# Matrix Types

- ▶ **Matrices** feature heavily in MATLAB code.
- ▶ LABMATE supports **type annotations** for matrices:

```
%> A :: [ 1 x 2 ] int
```

```
A = [ 3 4 ]
```

```
%> B :: [ 2 x 4 ] int
```

```
B = [ 1 1 1 1  
      5 6 7 8 ]
```

```
C = A * B
```

- ▶ We can ask for type information:

```
%> typeof C
```

```
%< C :: [Matrix 1 4 int]
```

- ▶ ...and can easily spot incompatible sizing:

```
D = B * A
```

```
%> typeof D
```

```
%< The expression D is quite a puzzle
```

LabMate can point  
out an error with D

# Matrix Types

- ▶ LABMATE processes arbitrary MATLAB code.
  - ▶ LABMATE does not rely on constants, **variables** work as well.

# Matrix Types

- ▶ LABMATE processes arbitrary MATLAB code.
  - ▶ LABMATE does not rely on constants, **variables** work as well.

```
function B = f(A)
    %> B :: [ 1 x 3 ] int
    B = [ 1 A ]

    %> typeof A
    %< A :: [Matrix 1 2 int]
end

A = 'hello'
%> typeof A
%< A :: string
```

# Matrix Types

- ▶ LABMATE processes arbitrary MATLAB code.
  - ▶ LABMATE does not rely on constants, **variables** work as well.

```
function B = f(A)
    %> B :: [ 1 x 3 ] int
    B = [ 1 A ]

    %> typeof A
    %< A :: [Matrix 1 2 int]
end

A = 'hello'
%> typeof A
%< A :: string
```



# Matrix Types

- ▶ LABMATE processes arbitrary MATLAB code.
  - ▶ LABMATE does not rely on constants, **variables** work as well.

```
function B = f(A)
    %> B :: [ 1 x 3 ] int
    B = [ 1 A ]

    %> typeof A
    %< A :: [Matrix 1 2 int]
end
```

LabMate infers type of A  
from the annotation on B

```
A = 'hello'
%> typeof A
%< A :: string
```

# Matrix Types

- ▶ LABMATE processes arbitrary MATLAB code.
  - ▶ LABMATE does not rely on constants, **variables** work as well.

```
function B = f(A)
    %> B :: [ 1 x 3 ] int
    B = [ 1 A ]

    %> typeof A
    %< A :: [Matrix 1 2 int]
end
```

```
A = 'hello'
```

```
%> typeof A
```

```
%< A :: string
```

tracks Matlab scope

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time  
%> unit kg :: Q({ `Mass })
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary

quantities  
define some base  
set of dimensions

```
%> dimensions V for Q over `Mass`, `Time`  
%> unit kg :: Q({ `Mass` })
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities and a canonical unit of measure
- ```
%> dimensions V for Q over `Mass`, `Time`  
%> unit kg :: Q({ `Mass` })
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

can be arbitrary group  
expression over V

```
%> dimensions V for Q over `Mass, `Time  
%> unit kg :: Q({ `Mass })
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time  
%> unit kg :: Q({ `Mass })
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time  
%> unit kg :: Q({ `Mass })  
  
%<{  
kg = 1;  
%<}
```



# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time
```

```
%> unit kg :: Q({ `Mass })
```

```
%<{
```

```
kg = 1;
```

```
%<}
```

this is a “magic” response  
that LabMate emits

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time
```

```
%> unit kg :: Q({ `Mass })
```

```
%<{
```

```
kg = 1;
```

```
%<}
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time  
%> unit kg :: Q({ `Mass })
```

```
%<{  
kg = 1;  
%<}
```

- ▶ We can then use the unit of measure for quantities:

```
y = 5*kg  
%> typeof y  
%< y :: Quantity (Enum [`Mass, `Time])  
%<                               {`Mass}
```

# Dimensions and Quantities

- ▶ LABMATE has support for arbitrary quantities.

```
%> dimensions V for Q over `Mass, `Time
%> unit kg :: Q({ `Mass })

%<{
kg = 1;
%<}
```

- ▶ We can then use the **unit of measure** for quantities:

```
y = 5*kg
%> typeof y
%< y :: Quantity (Enum [`Mass, `Time])
%<                               {`Mass}
```

turn a value of a dimension-  
less type into a quantity

# Dimensional Consistency for Matrices

- ▶ Most MATLAB programs **do not work on uniform matrices**: type of the entry  $e_{i,j}$  might depend on the indices  $i$  and  $j$ .

# Dimensional Consistency for Matrices

- ▶ Most MATLAB programs **do not work on uniform matrices**: type of the entry  $e_{i,j}$  might depend on the indices  $i$  and  $j$ .
- ▶ A common scenario when working with matrices of quantities:

```
%> dimensions V for Q over `L, `M, `T  
%> unit metre :: Q({ `L })  
%> unit kg :: Q({ `M })  
%> unit sec :: Q({ `T })
```

# Dimensional Consistency for Matrices

- ▶ Most MATLAB programs **do not work on uniform matrices**: type of the entry  $e_{i,j}$  might depend on the indices  $i$  and  $j$ .
- ▶ A common scenario when working with matrices of quantities:

```
%> dimensions V for Q over `L, `M, `T
%> unit metre :: Q({ `L })
%> unit kg :: Q({ `M })
%> unit sec :: Q({ `T })
```

- ▶ Work in progress: LABMATE support for such matrices

```
% > A :: [ i <- [{ } {`T}]
%           x j <- [{ } {`L}]
%           ] Q({`M * j / i})
A = [ 2*kg          5*kg*metre
      3*kg/sec      4*kg*metre/sec ]
```

## Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.



# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.

# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.
  - ▶ Not every annotation is needed; document the interesting ones.

# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.
  - ▶ Not every annotation is needed; document the interesting ones.
- ▶ MATLAB expressions are translated to LABMATE internal **core type theory**.

# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.
  - ▶ Not every annotation is needed; document the interesting ones.
- ▶ MATLAB expressions are translated to LABMATE internal **core type theory**.
  - ▶ Matrix types are parametrised over 5 parameters with dependencies between them.

# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.
  - ▶ Not every annotation is needed; document the interesting ones.
- ▶ MATLAB expressions are translated to LABMATE internal **core type theory**.
  - ▶ Matrix types are parametrised over 5 parameters with dependencies between them.
  - ▶ Quantities are modelled as the free Abelian group over a base set of dimensions.

# Implementation Details

- ▶ MATLAB programs are modelled as **trees of commands**, rather than sequence of commands.
  - ▶ Type information is propagated (consistently) throughout the tree; can put type annotations after variable declarations.
  - ▶ Not every annotation is needed; document the interesting ones.
- ▶ MATLAB expressions are translated to LABMATE internal **core type theory**.
  - ▶ Matrix types are parametrised over 5 parameters with dependencies between them.
  - ▶ Quantities are modelled as the free Abelian group over a base set of dimensions.
  - ▶ The typechecker understands **nontrivial algebraic properties**.

## Current Progress & Future Plans

- ▶ LABMATE is under active development.
  - ▶ Available on GitHub, please get in touch if interested.

# Current Progress & Future Plans

- ▶ LABMATE is under active development.
  - ▶ Available on GitHub, please get in touch if interested.
- ▶ Work in the pipeline:
  - ▶ **Uniqueness of representation**: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking.



# Current Progress & Future Plans

- ▶ LABMATE is under active development.
  - ▶ Available on GitHub, please get in touch if interested.
- ▶ Work in the pipeline:
  - ▶ **Uniqueness of representation**: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking.
  - ▶ **Quality of life improvements**: better messages and more readable responses from LABMATE.

# Current Progress & Future Plans

- ▶ LABMATE is under active development.
  - ▶ Available on GitHub, please get in touch if interested.
- ▶ Work in the pipeline:
  - ▶ **Uniqueness of representation**: currently, a matrix with quantities can have more than one corresponding type; this might lead to odd behaviour during typechecking.
  - ▶ **Quality of life improvements**: better messages and more readable responses from LABMATE.
- ▶ We want to extend our coverage to loops and conditionals in the future.