equationReader extension

equationReader has been publicly released on github at https://github.com/Marupio/equationReader/
Refer to this website for the latest documentation and source code.

1 Introduction

Latest version 0.6.0, released August 29, 2013. Known to work with:

- OpenFOAM 1.6-ext
- OpenFOAM 1.7.x
- OpenFOAM 2.0.x
- OpenFOAM 2.1.x
- OpenFOAM 2.2.x

1.1 What is it?

equationReader is an extension to OpenFOAM that allows you to work with user-entered equations. For example:

```
U.x         "sin(pi_ * t / 4)";
U.y         "rho * nut / L";
U.z         0;
nu          nu [0 2 -1 0 0 0 0] "1.2 + 3 * alpha^sin(pi_/6)";
aScalar     "nu / max(5, alpha)";
alpha       1.3;
```

1.2 What it isn't

Installing **equationReader** will not give any existing solvers the capability to read equations from dictionaries. It will only work on custom solvers that specifically include the **equationReader** in their inner-workings.

In the future, I plan to create a branch of OpenFOAM® that integrates **equationReader** directly into its core libraries, thus giving every existing solver equation-reading capabilities.

1.3 Features

- **Works with most fields** - Now works with single elements, fields, DimensionedFields and GeometricFields;
- **Works with most Types** - Now works with scalars, vectors, and all kinds of tensors;
- **Flexible data sources** - In addition to these types, equations can also lookup values from dictionaries, and you can create anactiveVariable that derives its value on-the-fly.
- **Order of operations** - it is fully compliant with the conventional order of operations to an arbitrary parenthesis depth;
- **Equation dependency tracking** - equations can depend on one another to an arbitrary hierarchy depth;
- **Circular-reference detection** - it will halt computations when a circular reference is detected;
- **On-the-fly equation mapping** - it will automatically perform substitution on other equations when they are needed, even if they aren't specifically called for in the solver; and
- **Dimension-checking** - fully utilizes OpenFOAM's built-in dimension-checking, or you can force the outcome to a specific dimension-set to quickly disable it (if you are lazy).

Limitations: Although **equationReader** works with all Types, at its core, it is just a scalar calculator with dimensions. Therefore, you can't use vector operators, let alone tensor operators. Each equation must be expressed in scalar components.

1.4 Why would you need this?

Let the user define their own equations - this makes your application **more user-friendly**, and **more flexible**. But don't reinvent the wheel - if you are only working with *boundary conditions* or *initial conditions*, Bernhard's swak4Foam would probably be more suitable.

1.5 Update Info

- *2010-07-21*: Initial release
- *2010-08-05*: Bug-fix - differentiated versions for OpenFOAM-1.5.x/1.5-dev and OpenFOAM-1.6.x+
- *2010-08-12*: Major upgrade
    - Introducing `IOEquationReader` - EquationReader is now an `IOobject`. This enables automatic output
    - Added support for `scalarList` data sources - including `scalarField`, `volScalarField`, etc.
    - Removed the need for pointers for data sources
    - Cleaned up available functions
- *2010-10-16*: Bug fixes and added full support for fields
- *2011-04-06*: Major upgrade
    - Efficiency improvement - pointer functions have been implemented to increase computation speed by an order of magnitude (at least).
    - Improved dimension-checking on all functions.
    - Added a fieldEvaluate function for active equations whose output is to a scalar field.
    - Bug fix to get it working with 1.6-ext and higher.
- *2011-09-13:* Major upgrade

- ***Now a stand-alone library***.
- Now works with ***vectors and tensors***:
  - scalar;
  - vector;
  - tensor;
  - diagTensor;
  - symmTensor; and
  - sphericalTensor.
- ***Now works with GeometricFields***
- Dimension checking is now performed separately, improving efficiency of field and GeometricField calculations.
- Interface changes:
  - Add data functions reorganized / changed.
  - Evaluate functions reorganized / changed.
  - Update functions removed.

- *2011-09-25:* ***Version 0.5.0***
  - Improved treatment of fields - ***now approximately 10x faster***.
  - Introduced version numbers to keep track of changes.
- *2012-10-25:* **Version 0.5.1**
  - Moved to git
  - Bug fixes:
    - Circular reference detection now working
- *2013-08-29:* **Version 0.6.0**
  - Uploaded to github and OpenFOAM-extend

- Restructured applications and tutorials directories for consistency
- Made opening splash optional

2 Efficiency

2.1 How fast is equationReader?

The most recent version of **equationReader** (*Version 0.5.0*, released September 25th, 2011) handles fields roughly 10x faster than the previous version. Overall, **equationReader** now takes approximately 5.87 times longer than a hard-coded solution when handling`GeometricFields`. That's for a simple equation. For more complex equations, **equationReader**'s performance improves.

Straight up `scalars` are still much slower. I haven't benchmarked the latest version, but previous versions were coming in at around 300 x slower.

2.2 Will it get faster?

Yes! The next plan is to have **equationReader** *compile* your equations at runtime. In theory, they will execute as fast as a hard-coded solution, less a small amount of overhead with the function call.

2.3 Parsing and evaluating

There is a difference between *parsing* and *evaluating*. When the equation is first read, it is a *human-readable* string expression.**equationReader** translates the *human-readable* form into an *operation list*. This is *parsing*. To calculate the result, **equationReader** does a `forAll(operations, i)`. This is *evaluating*.

Parsing happens only once, and is slow. *Evaluating* happens at every cell index, at every timestep (or however you've used it), and it is fast.

3 Installation

3.1 OpenFOAM-extend

If you have a recent version of OpenFOAM-extend, you may already have **equationReader** installed.

Type this command:

```
[ -d "$WM_PROJECT_DIR/src/equationReader" ]&&echo "Yes"||echo "No"
```

If the response is "Yes" then you already have it.

3.2 Git installation

A git installation will allow you to download the latest updates to **equationReader**.

equationReader is *git-tracked* seperately from OpenFOAM, so if your OpenFOAM installation is also *git-tracked*, it is advisable to put it in a seperate directory. Alternatively, if you use the latest version of OpenFOAM-extend, **equationReader** is incorporated within the main git repository.

Therefore, choose a seperate directory, for example:

```
-OpenFOAM
 |-OpenFOAM-2.2.x
 | |-applications
 | |-src
 | '-etc, and so on
 '-equationReader
```

To duplicate the structure above:

```
cd $WM_PROJECT_DIR
cd ..
git clone https://github.com/Marupio/equationReader.git
```

NOTE: If you have OpenFOAM 1.7.x or earlier, at this point you need to edit the file

`src/equationReader/include/versionSpecific.H`:

Comment out the second line:

```
//#define ThisIsFoamVersion2
```

For any version of OpenFOAM, complete the installation:

```
cd equationReader/src/equationReader
wmake libso
cd ../../applications/solvers/equationReader/equationReaderDemo
wmake
```

To later update equationReader:

```
cd $WM_PROJECT_DIR
cd ..
git pull
cd equationReader/src/equationReader
wmake libso
cd ../../applications/solvers/equationReader/equationReaderDemo
wmake
```

3.3 Manual installation

To manually install equationReader:

1. Download the code:

   - get the latest code from https://github.com/Marupio/equationReader/archive/master.zip
   - or use the zip file stored on the website linked-to from the DOI.

2. Open a terminal window and browse to the folder with your download.

3. Execute the following commands. You should be able to just copy and paste all 8 lines into your terminal:

```
unzip equationReader-master.zip
mv equationReader-master/README equationReader-master/tutorials/equationReader
cp equationReader-master/* $WM_PROJECT_DIR
rm -rf equationReader-master
cd $WM_PROJECT_DIR/src/equationReader
wmake libso
cd $FOAM_APP/solvers/equationReader/equationReaderDemo
wmake
```

equationReader should now be installed.

4 Testing the installation

To test the installation, copy the new `tutorials/equationReader/` directory to your run directory, and run `equationReaderDemo`.

5 Using equationReader

5.1 Dictionary syntax

Any of these formats are acceptable to **equationReader**:

5.1.1 Standard equation

```
keyword    "equation";
keyword    scalar;
```

e.g.:

```
endTime    "2*pi_/360*60";
gamma      1.58e-6;
```

The **standard equation** format performs dimension checking for every operation. Use this if you want OpenFOAM to be strict about the dimensions you use. This has may unexpected consequences. For example:

- `sin(time)` is wrong because you can't have dimensions in any transcendental functions; and
- `max(deltaT, SMALL_)` is wrong because `SMALL` is dimensionless.

If this is too troublesome, you can also use:

5.1.2 Dimensioned equation

```
keyword    [dimensionSet] "equation";
keyword    [dimensionSet] scalar;
keyword    ignoredWord [dimensionSet] "equation";
keyword    ignoredWord [dimensionSet] scalar;
```

e.g.:

```
nu      [0 2 -1 0 0 0 0] "1 / (1e-5 + 2.3/4000 + SMALL_)";
rho     [1 -3 0 0 0 0 0] 1.235;
delta   delta [0 1 0 0 0 0 0] "sin(pi_ * t)";
alhpa   alpha [0 1 0 0 0 0 0] 3.2;
```

The **dimensioned equation** format disables dimension checking, and forces the final result to a given *dimensionSet*. Also note the optional *ignoredWord* - this allows **equationReader** to be compatible with *dimensionedScalar* formats.

5.2 Equation syntax

equationReader uses the conventional order of operations **BEDMAS**, then left to right:

- **B**rackets (and functions);
- **E**xponents;
- **DM** - division and multiplication; and
- **AS** - addition and subtraction.

It's just like Excel, except exponents 'a^b' don't work - use 'pow(a,b)' instead.

- you can use any amount of whitespace you want (use a backslash for a line break);
- multiplication is `*`, for example `2*3` is `6`;
- there is no implied multiplication - you must explicitly use `*`. For example:

  `2 sin(theta)` **INCORRECT**

  `2 * sin(theta)` **CORRECT**

  and

  `2(3 + 4)` **INCORRECT**

  `2 * (3 + 4)` **CORRECT**

5.2.1 Mathematical constants

equationReader recognizes all the mathematical constants I could find in the OpenFOAM library. To specify a mathematical constant, append the regular OpenFOAM format with an underscore '_'. The available constants are:

- `e_` (Euler's number);
- `pi_`;
- `twoPi_`;
- `piByTwo_`;
- `GREAT_`;
- `VGREAT_`;
- `ROOTVGREAT_`;
- `SMALL_`;
- `VSMALL_`; and
- `ROOTSMALL_`.

5.2.2 Functions

Functions available to **equationReader** are:

- pow(x)
- sign(x)
- pos(x)
- neg(x)
- mag(x)
- limit(x, y)
- minMod(x, y)

- sqrtSumSqr(x, y)
- sqr(x)
- pow3(x)
- pow4(x)
- pow5(x)
- pow6(x)
- inv(x)
- sqrt(x)
- cbrt(x)
- hypot(x, y)
- exp(x)
- log(x)
- log10(x)
- sin(x)
- cos(x)
- tan(x)
- asin(x)
- acos(x)
- atan(x)
- atan2(x, y)
- sinh(x)
- cosh(x)
- tanh(x)
- asinh(x)

- acosh(x)
- atanh(x)
- erf(x)
- erfc(x)
- lgamma(x)
- j0(x)
- j1(x)
- jn(x, y)
- y0(x)
- y1(x)
- yn(x, y)
- max(x, y)
- min(x, y)
- stabilise(x, y)

5.3 Troubleshooting your equations

Your equations may cause you trouble, such as:

- Giving you a `SIGFPE`; or
- Failing dimension checks.

If this happens and you don't know why, **equationReader** has a detailed set of debug switches to help.

To change the debug switch, edit the `OpenFOAM/etc/controlDict` file and add:

```
equationReader     integerValue;
```

to the `DebugSwitches` list.

The debug switches available are:

0. silent mode;

1. scalar logging (light);

2. scalar logging (verbose);

3. dimension logging (light);

4. dimension logging (verbose);

5. scalar & dimension logging (light); or

6. scalar & dimension logging (verbose).

The scalar logging will report scalar-related operations to the console. The dimension logging, relates to dimensionSet operations. *verbose* reports operation-by-operation, so it can be overwhelming.

6 Programming with equationReader

Most of the programming features can be gleaned from the **equationReader** demo application. Please also refer to that.

6.1 Creating an equationReader object

To add **equationReader** to an application:

- Put `#include "IOEquationReader.H"` at the top of your main source file;
- Put `#include "createEquationReader.H"` somewhere after `createTime`;

6.2 Adding data sources

You need to add data sources - this is where **equationReader** looks for its variables.

6.2.1 Beware of duplicate sources

Currently, **equationReader** does not check if you are adding multiple variables of the same name. When this happens, you never know which source will be used. I didn't add it because it didn't occur to me until I started writing this paragraph. Expect it in the future.

6.2.2 Is the data permanent?

Data must be permanently available. For instance, `mesh.C()` is a valid data source because it returns a &reference. But `turbulence->R()` is not valid because it returns an object (or `tmp<object>`), and hence is *derived* from other permanent sources.

To use *derived* data sources, there are two options:

1. Create a permanent copy, and update it at every timestep. This is demonstrated in the **equationReaderDemo** application.
2. Create an *activeVariable*.

6.2.3 Active variables (advanced developers)

An *activeVariable* is one that does not permanently store its data, and provides values *on-demand* to **equationReader**. The key to this is it must be able to calculate a single cell value on-demand, and not the entire field at once. The interface is given in the `equationReader/equationVariable/equationVariable.H` file.

6.2.4 Functions to add data sources

To add data sources:

- for `scalars`, `dimensionedScalars`, `scalarFields`, `GeometricScalarFields`, etc.:

```
eqns.scalarSources().addSource(scalarObject);
```

  or if it doesn't have its own name (i.e. `scalars` and `scalarFields`) or you want to assign it a different name:

```
eqns.scalarSources().addData(scalarObject, name);
```

- for `vectors`, `dimensionedVectors`, `vectorFields` `GeometricVectorFields`, etc.:

```
eqns.vectorSources().addSource(vectorObject);
```

or if it doesn't have its own name (i.e. `scalars` and `scalarFields`) or you want to assign it a different name:

```
eqns.vectorSources().addSource(vectorObject, name);
```

- and so on for other Types (`tensor`, `diagTensor`, `symmTensor`, and `sphericalTensor`);
- for `dictionaries` or `activeVariables`:

```
eqns.addSource(dataObject);
```

6.3 Reading in the equations

To read equations from a dictionary use:

```
eqns.readEquation(dictionaryName, equationName);
```

6.4 Searching the equations

equationReader allows you to search its equations. Similar to the `dictionary` object, this will return `true` if *`equationName`* exists:

```
eqns.found(equationName);
```

The *evaluate* functions below call for `eqnNameOrIndex`. This means you can either use a `word` (the `equationName`), or a `label` (the `equationIndex`). The `equationIndex` is faster, as **equationReader** doesn't have to perform its own lookup. ***Never assume the `equationIndex` is equal to the order in which the equations were read.*** If the equations depend on one another, they may not always be in the same index. To learn the `equationIndex`, use:

```
equationIndex = eqns.lookup(equationName);
```

6.5 Evaluating equations

Once you are done adding data sources, and reading equations, you can start evaluating equations.

6.5.1 All data sources required

When evaluating an equation, **equationReader** needs access to all the possible variables and other equations that equation might depend on. If that variable or equation isn't found, **equationReader**

produces a **FatalError**. ***Therefore it is a mistake to try adding more data sources after the first evaluation.***

6.5.2 No mesh available

equationReader doesn't care about the mesh... all it cares about are the sizes of the the fields. ***The size of the variable fields must match.*** Index checking is expensive, so it is only available in `FULLDEBUG` mode. These rules apply:

- a single-element variable (e.g. a `scalar`, or a `dimensionedVector`) is assumed *uniform throughout the entire domain*, and can be used in any equation;
- a field variable (e.g. a `scalarField`, or a `DimensionedVectorField`) does not have a boundary field, therefore it is only available to equations of other fields or internal fields. Attempting to use it in a `GeometricField` is a mistake; and
- a `GeometricField` variable can be used with any equation.

There are two indices to indicate field / boundary field position:

- `cellIndex` - this is the position within a field (e.g. cell number in the internal field, or face number on a boundary patch);
- `geoIndex`:
    - 0 = the internal field;
    - greater than 0 = the boundary patches. The `geoIndex` is therefore 1-indexed on the boundaryField: `patchI = geoIndex - 1`.

If you omit either of these in the evaluation equations, they are assumed equal to zero.

6.5.3 Evaluation functions

- for single element types:

```
scalarA = eqns.evaluateScalar
(
    eqnNameOrIndex,
```

```
    [cellIndex],
    [geoIndex]
);
vectorA.x() = eqns.evaluateScalar
(
    xEqnNameOrIndex,
    [cellIndex],
    [geoIndex]
); // and so on for all its components
tensorA.xx() = eqns.evaluateScalar
(
    xxEqnNameOrIndex,
    [cellIndex],
    [geoIndex]
); // and so on for all its components
```

- for `dimensionedScalars`:

```
dimensionedScalarA = eqns.evaluateDimensioned
(
    eqnNameOrIndex,
    [cellIndex],
    [geoIndex]
);
```

- for other `dimensionedTypes` - there is no elegant dimensionChecking... use this hack:

```
vectorA.x() = eqns.evaluateScalar
(
    xEqnNameOrIndex,
    [cellIndex],
    [geoIndex]
);
vectorA.y() = eqns.evaluateScalar
(
    yEqnNameOrIndex,
    [cellIndex],
    [geoIndex]
);
vectorA.z() = eqns.evaluateScalar
(
    zEqnNameOrIndex,
    [cellIndex],
    [geoIndex]
);
vectorA.dimensions() = eqns.evaluateDimensions(xEqnNameOrIndex);
vectorA.dimensions() = eqns.evaluateDimensions(yEqnNameOrIndex);
vectorA.dimensions() = eqns.evaluateDimensions(zEqnNameOrIndex);
```

- for `scalarFields`:

```
eqns.evaluateScalarField(resultScalarField, eqnNameOrIndex, [geoIndex]);
```

or

```
eqns.evaluateTypeField
(
    resultScalarField,
    dummyWord,
    eqnNameOrIndex,
    [geoIndex]
);
```

- for `vectorFields`:

```
eqns.evaluateTypeField
```

```
(
    resultVectorField,
    "x", // this is the component name
    xEqnNameOrIndex,
    [geoIndex]
); // and so on for the "y" and "z" components
```

- and so on for other `typeFields`;

- for `DimensionedScalarFields`:

```
eqns.evaluateDimensionedScalarField
(
    resultDimensionedScalarField,
    eqnNameOrIndex,
    [geoIndex]
);
```

do not use `evaluateDimensionedTypeField` - this will fail for scalars;

- for `DimensionedVectorFields`:

```
eqns.evaluateDimensionedTypeField
(
    resultDimensionedVectorField,
    xEqnNameOrIndex,
    "x",
    [geoIndex]
); // and so on for the "y" and "z" components
```

- and so on for other `DimensionedTypeFields`;

- for `GeometricScalarFields`:

```
eqns.evaluateGeometricScalarField
(
    resultGeometricScalarField,
    eqnNameOrIndex
);
```

do not use `evaluateGeometricTypeField` - this will fail for scalars;

- for `GeometricVectorFields`:

```
eqns.evaluateGeometricTypeField
(
    resultGeometricTypeField,
    "x",
    xEqnNameOrIndex
); // and so on for the "y" and "z" components
```

- and so on for other `GeometricTypeFields`;

7 Uninstallation

7.1 The stand-alone (new) version

7.1.1 Am I running the stand-alone version?

Does `src/equationReader/` exist? If so, then you have the stand-alone version. You can check by entering the following command:

```
[ -d "$WM_PROJECT_DIR/src/equationReader" ]&&echo "Yes"||echo "No"
```

If it says Yes then you have the stand-alone version.

7.1.2 How do I uninstall the stand-alone version?

Enter the following commands into your console:

```
rm -rf $WM_PROJECT_DIR/src/equationReader
rm -rf $WM_PROJECT_DIR/applications/solvers/equationReader
rm -rf $WM_PROJECT_DIR/tutorials/equationReader
```

The stand-alone version of **equationReader** has been uninstalled.

7.2 The integrated (old) version

Since the integrated version of **equationReader** is compiled into the core of OpenFOAM, uninstallation requires file editting and recompiling of OpenFOAM.so.

7.2.1 Am I running the integrated version?

Does `src/OpenFOAM/db/dictionary/equation/` exist? If so, then you have the integrated version.

You can check by entering the following command:

```
[ -d "$WM_PROJECT_DIR/src/OpenFOAM/db/dictionary/equation" ]&&echo "Yes"||echo "No"
```

If it says Yes then you have the integrated version.

7.2.2 How do I uninstall the integrated version?

To uninstall the integrated **equationReader**:

1. Edit the `src/OpenFOAM/Make/files` file:

   Find and delete the bold lines below:

```
functionEntries = $(dictionary)/functionEntries
$(functionEntries)/functionEntry/functionEntry.C
$(functionEntries)/includeEntry/includeEntry.C
$(functionEntries)/includeIfPresentEntry/includeIfPresentEntry.C
$(functionEntries)/inputModeEntry/inputModeEntry.C
$(functionEntries)/removeEntry/removeEntry.C

equation = $(dictionary)/equation
$(equation)/equationReader/equationReader.C
$(equation)/equationReader/equationReaderIO.C
$(equation)/equation/equation.C
$(equation)/equation/equationIO.C
$(equation)/equationOperation/equationOperation.C

IOEquationReader = db/IOobjects/IOEquationReader
$(IOEquationReader)/IOEquationReader.C
$(IOEquationReader)/IOEquationReaderIO.C

IOdictionary = db/IOobjects/IOdictionary
$(IOdictionary)/IOdictionary.C
$(IOdictionary)/IOdictionaryIO.C
```

2. Edit the `src/OpenFOAM/primitives/Scalar/Scalar.C` file:

   Near the top, delete this line:

```
#include "equationReader.H"
```

   Near line 84, delete the bold section below:

```
    if (t.isNumber())
    {
        s = t.number();
    }
    else if (t.isString())
    {
        // DLFG 2010-07-21 Modifications for equationReader
        equationReader eqn;
        eqn.readEquation
        (
            equation
            (
                "fromScalar",
                t.stringToken()
            )
        );
        s = eqn.evaluate(0).value();
    }
    else
    {
        is.setBad();
        FatalIOErrorIn("operator>>(Istream&, Scalar&)", is)
            << "wrong token type - expected Scalar found " << t.info()
            << exit(FatalIOError);

        return is;
    }
```

From the terminal, enter the following commands:

```
rm -rf $WM_PROJECT_DIR/src/OpenFOAM/db/dictionary/equation
rm -rf $WM_PROJECT_DIR/src/OpenFOAM/db/IOobjects/IOEquationReader
rm -rf $WM_PROJECT_DIR/src/OpenFOAM/lnInclude
rm -rf $WM_PROJECT_DIR/applications/solvers/equationReader
rm -rf $WM_PROJECT_DIR/tutorials/equationReader
rm $FOAM_USER_APPBIN/equationReader*
rm $FOAM_APPBIN/equationReader*
cd $WM_PROJECT_DIR/src/OpenFOAM
rmdepall
wmake libso
```

The integrated version of **equationReader** has been uninstalled.