AN EQUATION EDITOR FOR MULTIDIMENSIONAL SCIENTIFIC DATA

Andrew Brown

Department of Electrical and Computer Engineering
University of Auckland, Auckland, New Zealand

Abstract

The Equation Editor for Multidimensional Scientific Data is a tool for the exploration of scientific datasets and transformations. It allows uses to create and edit equations as well as evaluate them. There are many other applications with some similar functionality, but none contain all the functionality that is required of the Multidimensional Equation Editor.

The user interface for the Equation Editor is designed to be easy to use and intuitive, using lessons learned by examining applications like Microsoft Visio[1]. The Equation Editor was also designed to be very extensible. This is achieved through the Equation Elements, which are a collection of operators and functions used in the equations. Simply by adding new Equation Elements more operators and functions become available to be used in the equations.

The Equation Editor was implemented using C++ and OpenGL, to make it multi platform and easy to support visualisation.

Although the Equation Editor met most of its goals and requirements it did not meet all of them. A key requirement, visualisation support was left out due to lack of time to implement this feature.

1. Introduction

The Equation Editor for Multidimensional Scientific Data is a tool designed to make it easy to explore multidimensional scientific data sets and the transformations that can be performed on them. It is similar to Microsoft Equation Editor[2] in that it allows you to create and edit scientific equations. However the Equation Editor will also evaluate the equations the user create with it and is designed to be included as part of a visualisation toolkit, functionality not included in the Microsoft package.

The equation editor is a tool that can be used by anyone with a need to edit equations, but it will be most useful to those who frequently deal with multidimensional scientific data sets. Those who specifically deal with these data sets for the purpose of scientific visualisation will find the tool most useful as it will make it easy to explore the different ways of visualizing the raw data. Visualisation is also the field that the tool was primarily designed for.

A multidimensional scientific data set is a multidimensional matrix whose fields can be scalars, vectors, tensors, or any other data type. These data sets are more commonly known as scalar, vector or tensor fields.

2. Other Equation Editing Packages

There are many different equation editing packages available that offer a lot of the functionality that the Multidimensional Equation Editor will provide. Some of the most common of these packages include; Microsoft Equation Editor, Tex/Latex[3], Matlab[4] and MathCAD[5]. The range of functionality these packages implement is vast, ranging from simple typesetting of equations to powerful equation solving and analysis tools.

Microsoft Equation Editor is the equation typesetting tool that comes with Microsoft Office. It allows you to create nicely typeset equations to be included in documents. Its menu system is for selecting operators is a little cumbersome to use and it has no support for evaluation of equations.

Tex/Latex is an equation typesetting tool quite similar to Microsoft Equation Editor, except that it is more powerful. Instead of using a menu system to create your equation the user type out a text description of the equation in the Tex format and the Tex/Latex engine will output the postscript code for the properly typeset. Like Microsoft Equation Editor, Tex/Latex has no support for the evaluation of the equations.

Matlab is a very powerful piece of mathematical software. Its programmatic interface is a very powerful way of representing equations, and it also provides an extensive collection of equation evaluation and solving tools. However the Matlab programmatic interface can be very difficult to master by those with little or no programming experience. It also has no support for the typesetting of equations.

MathCAD is another powerful mathematical software package. Like Matlab it has excellent support for evaluation of equations. It also lets you edit equations in a typeset view like Microsoft Equation Editor.

None of the above equation editing packages supported all the features that were desired in the Multidimensional Equation Editor. Some were missing support for the evaluation of equations, and none of these packages supported the tree view that was wanted in the Multidimensional Equation Editor.

3. Goals for the Equation Editor

The main goal for the Equation Editor was to develop a powerful tool for creating and editing equations. It had to support equation evaluation and make it easy to explore datasets and transformations through an easy to use interface and use of visualisations. The Equation Editor was envisaged as be a tool that is desirable to include as part of a visualisation toolkit.

It would also be desirable to make the Equation Editor portable.

4. Design

4.1. Requirements

Functional Requirements

- Facilitate the creation of equations and transformations on multidimensional scientific data.
- Ability to evaluate equations.
- Ability to work with multiple different data types, like scalar, vector and tensor fields.
- Extensible so it is easy to add more operations and functions to the equations.
- Include multiple different views of the equation including a typeset view (the normal view you have of an equation when it is printed, see Figure 2.) and a tree view.
- Allow editing of the equations in all views and keep the views synchronised so that changes that get made in one view are immediately reflected in the other views.
- An optional requirement was to include support for visualisation, as both a visualisation of the resulting transformed data that is output from the equations as well and extending the tree view to show a visualisation of the data at each node in the tree.

Non-functional Requirements

- Intuitive easy to use user interface
- Easily portable to other platforms

4.2. User Interface Considerations

In designing the user interface one of the primary concerns was creating an interface that is intuitive and easy to use. To do this it was decided that that the best way to go about doing this was to leverage some of the interface features out of other programs. The program that that most influenced the equation editor was Microsoft Visio. The equation editor like Visio has a multiple page toolbar on the left hand side of the window. The toolbar contains a lists of the of the equation operators that can be used in the equations. The operators are sorted into pages that categorise the sorts of data the operators operate on (i.e. all operators that operate on scalars are on one page of the toolbar, while all operators that operate on vectors are on another).

It was also decided to make the interface primarily drag and drop. This is because most people are familiar with the drag and drop interface as Windows makes much use of it. Drag and drop is also used a lot in Visio. With much of the Equation Editor's interface being similar to Visio the Equation editor should seem familiar to those who have used Visio, and for those who have not the Equation Editor should be very easy to pick up and start using like Visio is.

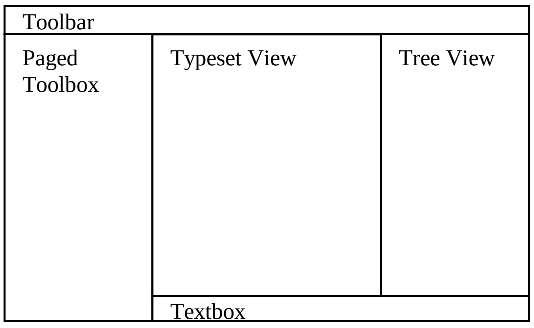
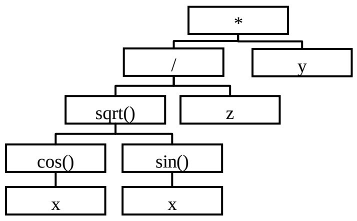


Figure 1 - User interface layout diagram

The main part of the window is divided into two different equation views with a text box below them. Figure 1 shows this layout. The left view is used for the typeset view and the right view for the tree view. The typeset view displays a list of equations laid out like how would normally write an equation. On the right of the typeset view is the tree view that displays the equation in a tree layout. The tree layout shows the exact order of the operations the data will undergo when the equation gets evaluated. Operations occurring first are further down the tree and the last operation is at the root of the tree. In the textbox below the other two views, the user can create an equation by typing out the equation in a format that is similar to that of Microsoft Excel[6]. The user can also create part of an equation and drag it from the text box to the required place in the equation they are working on. Figure 2 shows the differences between the typeset view, the tree view and the text version of the equation.



Tree View Layout

$$\frac{\sqrt{\sin(x)+\cos(x)}}{y} * z$$

Typeset View Layout

sqrt(sin(x) + cos(x))/y * z

Text Layout

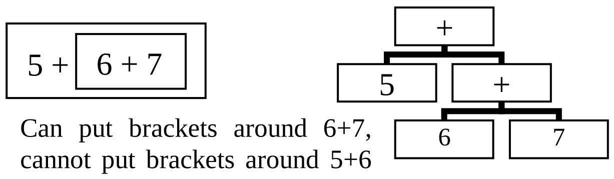
Figure 2 - The differences between the layouts

Some informal user testing was done on the interface of the equation editor. The feed back received from that was very positive. However it did highlight some flaws with the way in which the typeset view worked. In the evaluation version when dragging operators or equations around on the typeset view ant then dropping it on a division operator, the equation would rearrange itself. Users found this very off-putting as they had to familiarise them selves with the equations new layout before continuing to edit the equation. To solve this issue we had to redesign the data structures that backed up the different views, so that the equations could be edited as expected.

4.3. System Architecture

The Equation Editor went through two major changes in architecture. The first version of the architecture was fairly simple. It split the project into four parts the windowing system, the equation editor and the parser and the equation evaluator. The windowing system is where all the user interface functionality was to be implemented. The equation editor part was the part that handled everything to do with editing equations, including the drag and drop interface, the rendering of the equations in all views and the type checking of the equations to make sure the equations were correct. The parser is the part of the programme that handles the conversion form string form to the tree data structure that the equation editor part uses. The final part of the programme is the equation evaluator which uses the parser to read in the text version of the equations and then evaluated them. It kept as separate from the rest of the project as possible so that it could be built as a command line tool and used for batch processing of the equations. It was also desirable to keep it separate form the rest of the project to make it possible to develop the programme to use a third party tool for the evaluation of the equations.

However about halfway through the project it was found that this architecture was insufficient for the to meet the requirements for the Equation Editor. As the architecture originally had both the typeset and tree view's to use the same backing data structure. There were restrictions on how the user could edit data. Figure 3 shows an example of how editing was restricted. After some informal user testing highlighted this deficiency in how the data can be edited the decision was made to change the architecture.



Can put brackets around 6+7, cannot put brackets around 5+6 as they are not children of the same node.

Figure 3 - Equation editing restrictions

The new architecture kept the same windowing system as the original, large modifications were made to the rest of the programme. To fix the problem with editing the equations in the typeset view a new data structure was needed to back the view. The problem with the original tree was it was a very deep tree, with every operation increasing the depth of the tree. As the editor restricted you to only editing data at one node in the tree, it was very difficult to do things like inserting brackets in the correct location in the tree. The editor was therefore split into two, one part for each of the typeset and tree views. The typeset view was given a new data structure, that was designed to be as flat as possible, only branching where necessary mostly at brackets, exponents and when the division operation was encountered. Splitting the editor into two added another complexity to the project. Both views now needed to be able to communicate with each other to inform the other view if changes had been made to the equation. It was decided the easiest way to do this was to create a new equation parser for the typeset view and use a serialisation interface to update the other view. It was also decided to remove the type checking information form the editor and move it into the evaluator as that also allowed more flexibility in how the equations are created. This created an equation compilation stage that equations have to go through before the equations can be evaluated to check that they are correct. Figure 4 shows a diagram of the new architecture.

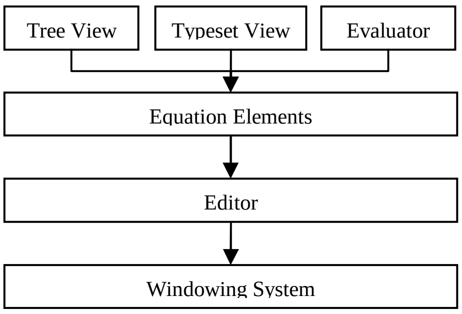


Figure 4 - Equation Editor Architecture

The main features of the architecture are described below:

Windowing System, responsible for the Equation Editor's GUI (Graphic User Interface), is platform independent and supports visualisation.

Editor, handles the editing of the equations. Implements features like the drag and drop interface.

Equation Elements, the operators and functions that are used to build up larger equations. Each operator contains all the information needed for the rendering, type checking and evaluating of the equations. The Equation Elements are the main way to keep the Equation Editor extensible. Just by adding more elements, the Equation Editor can be extended to support new functions and operation as well as new data types.

Tree View, controls the editing of the equations in the tree view window. Tells the windowing system how to draw the tree using information from the elements as well as the editor about where thing can be dragged and dropped in the tree view. It contains its own parser for serialising and de-serialising the equation. It is backed by a data tree that has the same structure as the tree view displays. It is structured so that every node is an operator and the operands are that nodes children. Figure 5 shows an example.

Typeset View. The typeset view is essentially the same as the tree view. It tells the editor how to render the equations in the typeset view window as well as how the equation can be edited, using information contained in the Equation Elements. It also implements a parser to serialise and de-serialise the equations. The backing data structure for the typeset view is quite different from the tree view in that the typeset view is a much flatter tree that puts as many nodes as possible on the same level only branching at brackets and at other times necessary. Figure 5 contains an example.

Evaluator. The Evaluator is the part of the programme that evaluates the equations. Like the tree view and typeset view it also contains its own parser. This enable the evaluator to be used as a standalone command line tool that can be used for batch processing of equations, which may be necessary if the data sets are very large. Its backing data structure is very similar to the tree used by the tree view, except it contains a lot of type checking information that it retrieves from the Equation Elements. This type checking data allows the evaluator to check the equations for errors before it begins evaluating them.

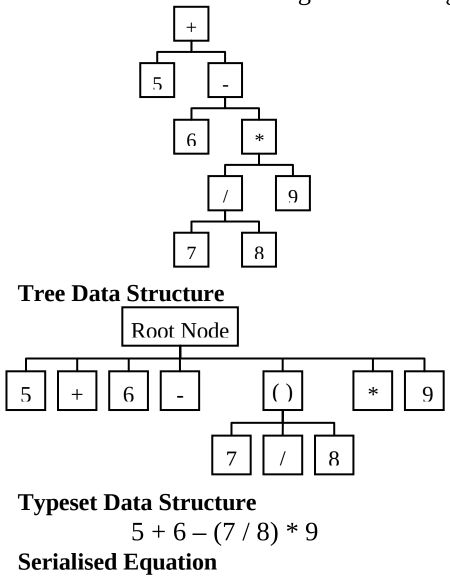


Figure 5 – Data structures diagram

5. Implementation

5.1. Platform

To meet the requirement for the project to be platform independent and to be able to support visualisation, it was decided to use ANSI C++ and OpenGL\GLUT[7] as the platform for the project. There are many other alternatives that could have been used for the platform. These included .NET, Java and C++ in combination with the windowing library TCL\TK[8]. In particular .NET was seriously considered due to its ease of use as a development platform, as well as its strong bindings to OpenGL using the Tao[9] libraries. However this option was rejected as there is currently only full support for it in Windows. Although .NET code can be run on Linux using the Mono[10] project, its implementation of the .NET libraries is not yet complete. Java was not used, even though it is the most platform independent alternative; due to its lack for support for three dimensional graphics making it not suitable for visualisation. The last option of C++ and TCL\TK was rejected because TCL\TK (a platform independent script based windowing library) required the instillation of the

TCL\TK libraries on machines that the code was to be used on. TCL\TK is also lacking in support for 3D graphics.

5.2. Windowing System

The windowing system used in the Equation Editor is a custom windowing library written on top of OpenGL and GLUT. Its design is quite similar to the design of the windowing system used in Java. It contains multiple controls including buttons, textboxes, scrollbars and the paged toolbar, used as the toolbar for the operators in the equation. The windowing system does not contain a complete set of controls as only controls that were needed were created.

All controls in the windowing system are rendered using OpenGL polygons. Each control contains a list of child controls allowing complex controls to be build up from a series of simple ones. To control the look and feel of the controls all of the rendering is done through a canvas object. The canvas object has a list of predefined colours and functions for rendering shapes and text. The canvas object also looks after tasks like making sure that nothing was drawn outside the bounds of the current control, which was very important when making the scrollable controls.

The windowing system also uses an event system. This system is very similar to the one used in Java and is responsible for the communication of information between the controls. The windowing system also contains the interface for the drag and drop system used for editing of the equations.

The windowing system currently has no support for visualisation. As visualisation was an optional feature it was left out due to time constraints. However due to the fact that the windowing system is built using OpenGL it should not be difficult to add support for visualisation. This is probably best achieved using a visualisation tool kit.

5.3. Editor

The editor is the main part of the Equation Editor. It combines the various other parts of the Equation Editor and joins them into one application. It takes the controls provided by the windowing system and combines them to create the Equation Editor's user interface (see Appendix A for a screen shot of the user interface).

5.4. Equation Elements

The Equation Elements represent the numbers, variable, operators and functions that can be used in the equations. They contain all the information needed by the Tree View and Typeset View for the layout and rendering of the equations. The Equation Elements also contain all the type information needed by the Evaluator for checking the equations for errors, as well as code needed for evaluation of the equations. This information includes the number of operands the Equation Element has. Scalars and variables have no operands, operators like addition and subtraction normally have two, and function like sine and cosine can have any number of operands.

The Equation Elements also contain information about how the operands are laid out. The can be next to each other like in multiplication, or one above the other like in division, or offset for exponents. Currently there are elements for most scalar operations as well as a few vector operations like dot a cross products.

5.5. Tree View

The Tree View is the part of the Equation Editor that renders the equations in a tree layout. It extends the functionality of the of the Windowing System by creating a new control, the Tree View Control, that displays the Tree View and allows users to edit the equations in that view. It has it own parser for converting equations to and from a serialised text form. The parsed serialised text form of the equation is used as the method for communication between this view, the Typeset View and the Equation Evaluator.

5.6. Typeset View

The Typeset View is much the same as the Tree View except it displays the equations differently, allows for some slightly different methods for editing the equations and is backed by a much flatter data structure than what is used by the Tree View, see Figure [ref].

5.7. Equation Evaluator

The Equation Evaluator takes the equations built in the editor and evaluates them. The Evaluator is also used by the Equation Editor to type check the equations and make sure they are mathematically correct. It is similar to the Tree and Typeset Views as it implements a parser to convert the equations for the serialised form provided by the Views and build a tree that can be used to evaluate the equation and type check the equations.

Due to time constraints only a simple evaluator was created that works on scalar values only, as a proof of concept. The evaluator would be best implemented using a separate mathematical package. This would allow a lot of operators and functions to be used in the equation without write code to evaluate the operation. Matlab was the package that was looked at for use in the Equations Editor, but was not used due to lack of time.

5.8. Parsers

There are three parsers used in the Equation Editor. These are used by both the Tree and Typeset View's and well as the Equation Evaluator. The parsers were created using Bison[11], and Flex[11], two very powerful tools that are used for creating parsers. Bison and Flex were chosen as the tool for crating the parsers as they are the two best known tools for the creating parsers. Other tools that were looked at were Yacc[11] and Lex[11] but these were not used as Bison and Flex are updated versions of these tools. Flex is a tool for creating a lexical analyser, a tool that takes a file and searches it for known tokens. Bison is a tool known as a ‘compiler compiler’, it creates a tool called a syntactical analyser that takes the list of tokens that the lexical analyser provides and turns them into an abstract syntax tree, based on the specified grammar. The abstract syntax tree is used to create the data structures that back the Tree View, the Typeset View and the Equation Evaluator.

Each of the three parsers that are used in the Equation Editor are different. The parser used in the Tree View is the simplest. It turns the serialised equation into the tree form displayed in the Tree View. The parser used in Typeset view is different to the one used in the Tree View as it is designed to produce a much flatter tree. The parser used by the Equation Editor is the most complex. It is very similar to the parser used by the Tree View except it performs type checking on the equations as well, using information from the Equation Elements.

6. Results

- The Equation Editor allows simple equations made of scalar and vector fields to be manipulated.
- The small group of users who tested the application found its interface to be quite intuitive and easy to use.
- The Equation Editor provides both the Tree and Typeset Views of the equation and allows the equation to be edited in these views. It also has a textbox where the serialised form of the equation can be edited.
- The Equation Editor does not currently contain a large number of operators. However the operators can easily be added to by adding new Equation Elements.
- The Equation Editor supports evaluation of the equations. This is currently limited to simple scalar equations, but could easily be extended through the use of a mathematic package like Matlab.
- The Equation Editor currently does not support visualisation. Support for visualisation should not be difficult to add by taking advantage of a visualisation library.

7. Future Work

There are many opportunities for Future Work on the Equation Editor. Some of these opportunities involve completion of features that were neglected due to lack of time as well as some new features that will make the Equation Editor a more powerful tool.

Visualisation. Support for visualisation would allow users of the Equation Editor to see the results of the transformation equations created in the editor. Visualisation could also be used to extend the Tree View, by putting a small visualisation at each node in the tree. These small visualisations would show how the data has been transformed to that point in the tree.

Equation Evaluation. The Equation Evaluator can be extended to take advantage of mathematical software packages like Matlab.

OLE Plug-in for Microsoft Office. To make it easier to use the equations created in the editor in documents, an OLE plug-in could be created. An OLE plug-in would allow equations it equations to easily be inserted into Office documents, making this Equation Editor an alternative to Microsoft Equation Editor. Another alternative to this would be to create a Tex/Latex output for the program.

Scripted Equation Elements. By including a scripting language in the Equation Editor, Equation Elements could be dynamically added to the editor. Currently to add new Equation Elements the program has to be recompiled.

8. Conclusions

The project met most of its goals and functional requirements. It is a useful and equation editing tool, although is functionality is limited due to features left out because of time constraints. It supports equation evaluation, and will be useful for exploring scientific datasets and transformations once more Equation Elements are created and the evaluation functionality is extended.

Due to lack of time support for visualisation could not be implemented and this restricts the Equation Editors usefulness as part of a visualisation toolkit.

Acknowledgements

I would like to acknowledge my supervisor Dr Burkhard Wuensche for his input and ideas that helped making this project successful. I would also like to acknowledge my project partner Andrew Llewelyn for his contribution to the project and for all his help and assistance on parts of the project I found difficult.

9. References

[1] “Microsoft Visio 2003”, [Online document] [cited 2005 September 2], Available HTTP:

http://office.microsoft.com/en-us/FX010857981033.aspx

[2] "Microsoft Equation Editor", [Online document] [cited 2005 May 5], Available HTTP: http://office.microsoft.com/en-us/assistance/HA011327531033.aspx

[3] "LaTeX", [Online document] [cited 2005 May 5], Available HTTP: http://www.latex-project.org/.

[4] "MATLAB", [Online document] [cited 2005 September 5], Available HTTP: http://www.mathworks.com/products/matlab/

[5] "MathCAD", [Online document] [cited 2005 May 5], Available HTTP: http://www.mathcad.com/

[6] "Microsoft Excel", [Online document] [cited 2005 September 5], Available HTTP: http://office.microsoft.com/en-us/FX010858001033.aspx

[7] "GLUT – The OpenGL Utility Toolkit", [Online document] [cited 2005 September 5], Available HTTP: http://www.opengl.org/resources/libraries/glut.html

[8] "Tcl\Tk", [Online document] [cited 2005 June 12], Available HTTP: http://www.tcl.tk/software/tcltk/

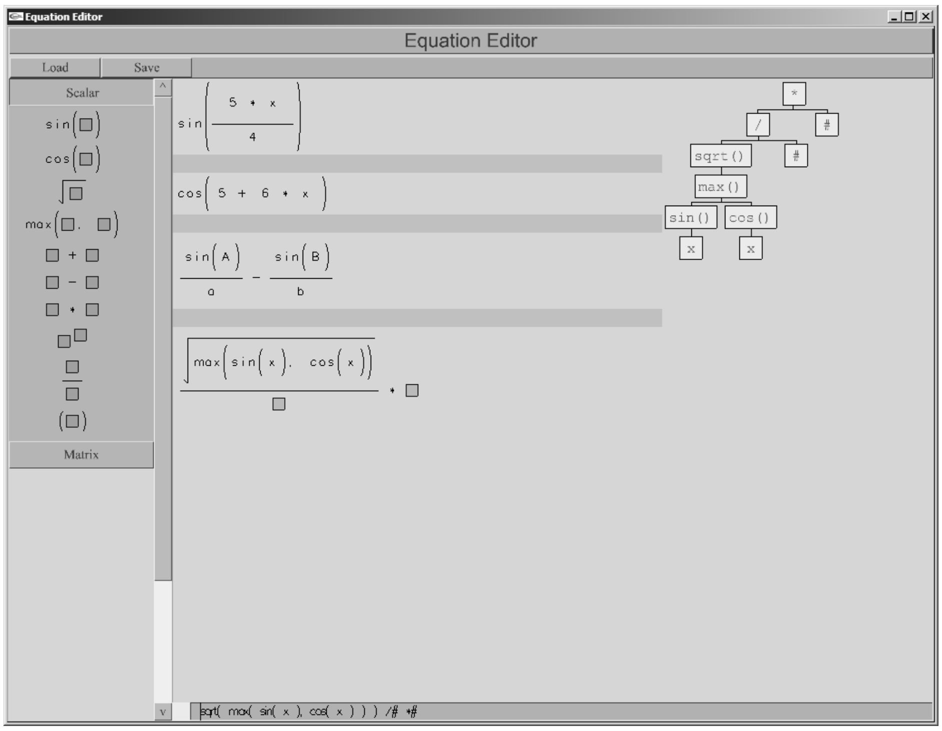
[9] "Tao", [Online document] [cited 2005 September 8], Available HTTP: http://www.mono-project.com/Tao

[10] "Mono", [Online document] [cited 2005 September 8], Available HTTP: http://www.mono-project.com

[11] "The LEX & YACC Page", [Online document] [cited 2005 September 8], Available HTTP: http://dinosaur.compilertools.net/

Appendix A

Screenshot of the Multidimensional Equation Editor



$$\sin\left(\frac{5 * x}{4}\right)$$

$$\cos\left(5 + 6 * x\right)$$

$$\frac{\sin(A)}{a} - \frac{\sin(B)}{b}$$

$$\frac{\sqrt{\max\left(\sin(x),\ \cos(x)\right)}}{\square} * \square$$

```
sqrt( max( sin( x ), cos( x ) ) ) /# *#
```