Connectivity in Maple 2022

\section*{Jupyter}

- The new \textit{Maple Kernel for Jupyter} is a program bundled with Maple which allows Maple to be used as the computation engine in a session of the Jupyter computation environment. Sessions are executed in a web browser and can be saved as notebooks combining explanatory text, mathematics, computations and media. These notebooks may be shared and used by other users of the Maple Kernel for Jupyter.

- To use the Maple Kernel for Jupyter, first ensure that Jupyter is installed on the same machine as your Maple installation. Then follow the instructions in \textit{Configuring the Maple Kernel for Jupyter} to make Maple available as a kernel within Jupyter.

- Once Maple is configured, it should appear within Jupyter as an available kernel type when creating a new document. In a notebook using Maple as a kernel, a code cell accepts input in standard \textit{Maple Notation} (also known as \textbf{1-D math}). Output is displayed using standard file formats supported by Jupyter such as \textit{LaTeX} and \textit{PNG}.

```
[1]: expand((x+y)^9);
1: x^9 + 9*x^8*y + 36*x^7*y^2 + 84*x^6*y^3 + 126*x^5*y^4 + 126*x^4*y^5 + 84*x^3*y^6 + 36*x^2*y^7 + 9*x*y^8 + y^9
[2]: diff(Out[1], x);
2: 9*x^8 + 72*x^7*y + 252*x^6*y^2 + 504*x^5*y^3 + 630*x^4*y^4 + 504*x^3*y^5 + 252*x^2*y^6 + 72*x*y^7 + 9*y^8
[24]: plot(x^4+x^3-7*x*x^2-x+6, view=[-4..3,-20..20], thickness=4, color="RoyalBlue")
```

\[ [-4, -3, -2, 0, 1, 2, 3] \]

\[ [-20, -10, 0, 10, 20] \]
**Jupyter Package**

The Jupyter package is a Maple package containing tools to support the Maple Kernel for Jupyter.

```maple
> with(Jupyter);

[CreateNotebook, ExtractCodeSources, GenerateKernelConfiguration, SetOutputRendererByType]
```

With the CreateNotebook command we can create a Jupyter notebook suitable for use with the Maple Kernel for Jupyter from a Maple help page, worksheet, or expression. Here we transform this help page into a Jupyter notebook in the current user's home directory.

```maple
> CreateNotebook( "Connectivity.ipynb", "updates,Maple2022, Connectivity", source=help, base=homedir );

20615
```

The Jupyter package also includes tools for generating configuration files to set up the Jupyter connection and controlling the way Maple output is displayed within Jupyter.

**The SMTLIB Package**

- The SMTLIB package has been augmented with a new command SMTLIB:-Session, which creates a session object. This purpose of this is to enable a persistent call to the underlying SMT engine, so that the initialization cost need be paid at most once per session.
- The Session object also maintains a stack of the solver state. Thus we can explore a particular subproblem of our current problem by pushing the stack with Push, performing a specialized query using Assert and Satisfy, and then popping with Pop to restore the prior state.
- First, we assert an equation and confirm it is satisfiable over the reals.

```maple
> s := SMTLIB[Session]();

SMTLIB Session

23844984
Variables: 0
Stack Depth: 0

> s:-Assert( x^2 - 25 = 0 ) assuming real;

> s:-Satisfy();

{x = 5}
```
• Now after pushing the session stack we impose another assumption, and discover that
the model has become unsatisfiable.

> s:-Push();
> s:-Assert( x > 5 );
> s:-Satisfiable();

false

> s:-Pop();
• After popping to restore the original problem, we can push again to explore the area to
the left of the known root, and find the one remaining real root.

> s:-Push();

1

> s:-Assert( x < 5 );
> s:-Satisfy();

{x = -5}

> s:-Pop();

0

▼ The DeepLearning Package

The **DeepLearning** package has undergone a number of updates and improvements in
Maple 2022.

▼ Indexing DeepLearning Tensors

**Tensor** objects in DeepLearning now accept indexing syntax similar to Matrices, Vectors,
and Arrays in Maple.

> restart:

> with(DeepLearning):

> M := RandomTensor( Gamma(2.5,3.3), [5,4,3], datatype=float[8],
seed=2022 );

```
M :=
[ DeepLearning Tensor
  Shape: [5, 4, 3]
  Data Type: float[8] ]
```
You can use this to create slices of an existing Tensor.

\[ M_2 := M[1..2,..,1..2]; \]

\[
M_2 := \begin{bmatrix}
\text{DeepLearning Tensor} \\
\text{Shape: [2, 4, 2]} \\
\text{Data Type: float[8]}
\end{bmatrix}
\]

Additionally we can use this to directly access scalar values from inside the Tensor.

\[ M[2,3,2]; \]

10.7615426222778

Alternatively the entire Tensor can be easily converted into a Vector, Matrix, or Array with the `convert` command.

\[ \text{convert} ( M, \text{Array} ); \]

\[
6.75313149056171 \\ 4.35116216422273 \\ 15.8836349575203 \\ 12.1491831021757 \\
11.7160640627332 \\ 2.03390958544607 \\ 9.96030379277387 \\ 9.36755377418358 \\
5.90730269483916 \\ 8.8558455121452 \\ 8.29299182444118 \\ 11.676676384969 \\
3.82283802557249 \\ 6.15935276225590 \\ 6.03589384550668 \\ 17.1526308248580 \\
12.9928253114120 \\ 3.36829965586089 \\ 8.30668703969549 \\ 14.8079805405892
\]

\(\text{slice of 5 } \times \text{ 4 } \times \text{ 3 Array}\)

\[ \text{Using GradientTape for Computing Tensor Gradients} \]

- A `GradientTape` is a execution context in which certain marked Tensors are watched for the purposes of computing their gradients.

\[ u := \text{Constant}([[3., 5.]], \text{datatype=float[8]}); \]

\[
u := \begin{bmatrix}
\text{DeepLearning Tensor} \\
\text{Shape: [1, 2]} \\
\text{Data Type: float[8]}
\end{bmatrix}
\]

- The following creates a tape object to track variables of interest.

\[ \text{tape} := \text{GradientTape}(); \]

\[
tape := \begin{bmatrix}
\text{DeepLearning GradientTape} \\
<\text{tensorflow.python.eager.backprop.GradientTape object at 0x0000022F309EC730}>\end{bmatrix}
\]

- The `Enter` and `Exit` commands activate and deactivate this context. Only operations which occur after `Enter` has been invoked are tracked.

\[ \text{Enter} ( \text{tape} ); \]
Convert between DeepLearning and Python objects

- As DeepLearning is built on the Google TensorFlow library using Maple's connectivity to Python, there is a close connection between many DeepLearning objects and TensorFlow objects. The convert command now permits easy conversion between DeepLearning objects and objects of type python from TensorFlow. This will simplify the adaptation of TensorFlow models to DeepLearning and facilitate the interaction of DeepLearning objects with TensorFlow features which have not been exposed in DeepLearning.

- The command convert(..., python) converts DeepLearning objects to their Python TensorFlow equivalents.

> t1 := Constant( <<<1,2,3>|<4,5,6>|<7,8,9>>, datatype=float[8] );

\[
t1 := \begin{bmatrix}
1 & 4 & 7 \\
2 & 5 & 8 \\
3 & 6 & 9
\end{bmatrix}
\]

> convert( t1, python );

"<Python object: tf.Tensor(
[[1. 4. 7.]
 [2. 5. 8.]
 [3. 6. 9.]], shape=(3, 3), dtype=float64)>"

> Python:-ImportModule("tensorflow as tf");
The command \texttt{convert(..., DeepLearning)} converts \texttt{DeepLearning} objects to their Python TensorFlow equivalents.

\begin{verbatim}
> pt2 := Python:-EvalString( "tf.random.uniform(shape=[3,4], maxval=20, dtype=tf.int32, seed=1701)" );
    pt2 := "<Python object: tf.Tensor(
        [[ 9  2  7 13]
        [ 3 18 12 11]
        [ 9 19 12 12]], shape=(3, 4), dtype=int32)"

> t2 := convert( pt2, DeepLearning );
    t2 := [DeepLearning Tensor
        Shape: [3, 4]
        Data Type: integer[4]]

> pt3 := Python:-EvalFunction( "tf.keras.layers.GaussianNoise", 5.5 );
    pt3 := "<Python object: <tensorflow.python.keras.layers.noise.GaussianNoise object at 0x0000022F309D56A0>"

> t3 := convert( pt3, DeepLearning );
    t3 := [DeepLearning Layer
        <tensorflow.python.keras.layers.noise.GaussianNoise object at 0x0000022F309D56A0>]
\end{verbatim}
\section*{Additional Updates}

\subsection*{Two-Dimensional Barcode Generation}

The \texttt{ImageTools:-GenerateBarcode} command can generate a 2-D barcode in the QR code format from an input string or ByteArray.

\begin{verbatim}
> with(ImageTools):

> qrCode := GenerateBarcode( "This is a test" );

> ImageTools:-Embed( Scale( qrCode, 9, method=nearest ) );
\end{verbatim}

\subsection*{Converting Data between Formats in Memory}

\begin{itemize}
\item The \texttt{convert} command to objects of type \texttt{Array}, \texttt{Matrix}, \texttt{Vector}, \texttt{string} and \texttt{ByteArray} has been supplemented with an additional option, \texttt{sourceformat}. This reads encoded data from a \texttt{string} or \texttt{ByteArray} in a specified to the named type with no need for external files.
\end{itemize}
Here we can read directly from a comma-delimited string to a Matrix:

```maple
c> convert( "1,2
3,4", Matrix, sourceformat="CSV" );

1 2
3 4
```

The format specified must be one accepted by the `Import` command. When the output type is not `string` or `ByteArray`, `sourceformat` may be abbreviated simply as `format`.

Additionally, the `convert` command to objects of type `string` and `ByteArray` has been supplemented with an additional option, `targetformat`. This writes the encoded data to the output string or ByteArray in the named format, which must be one accepted by the `Export` command.

This has the same effect as exporting the content to a file in the chosen format with `Export` and then reading it to a string or ByteArray with `FileTools:-Text:-ReadFile` or `FileTools:-Binary:-ReadFile` respectively, but is achieved without any user-visible need for file input or output.

When the input type is not `string` or `ByteArray`, `targetformat` may be abbreviated simply as `format`.

In these examples we convert expressions to strings encoded in `JSON` and `LaTeX` respectively.

```maple
c> T := table([ "firstname" = "G", "lastname" = "Raymond", "DOB" = "1960-02-28" ]); 

T := table([ "lastname" = "Raymond", "firstname" = "G", "DOB" = "1960-02-28" ])

c> convert( T, string, format="JSON" );

"{
   "DOB": "1960-02-28",
   "firstname": "G",
   "lastname": "Raymond"
}"

c> convert( { solve(a*x^2 + b*x + c = 0, x) }, string, format="LaTeX" );

"\left\{\frac{-b C\sqrt{-4 a c Cb^{2}}}{2 a}, -\frac{b C\sqrt{-4 a c Cb^{2}}}{2 a}\right\}"
```
• In the following example we convert a 3-D plot to a ByteArray encoded in the STL format

> knot := algcurves:-plot_knot((-x^7 + y^3)*(-2*x^5 + y^2), x, y, epsilon = 0.8, radius = 0.1, tubepoints = 9);

> convert( knot, ByteArray, targetformat="STL" );

Multipart Form Post

• The new MultipartFormPost command in the URL package provides another protocol for uploading data to a web service. This command mimics the behavior of a submit button on a form-based webpage; usually one asking for a file upload and other input data. The new command can package all of the entries together, send them to the website in the correct format (either for storage, or analysis), and get the post-submit page back for further processing if needed.