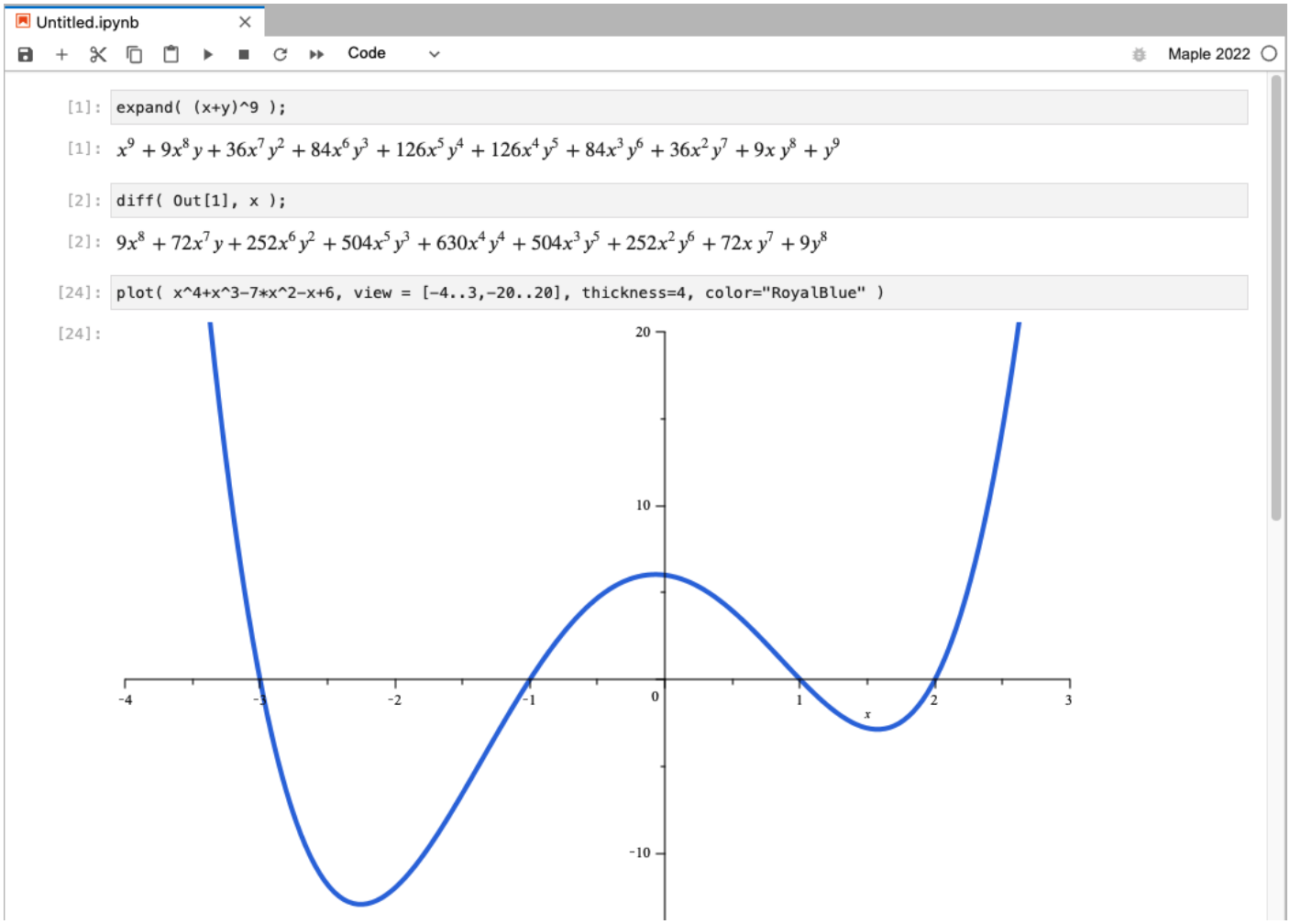
Connectivity in Maple 2022

Jupyter

- The new 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 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 Maple Notation (also known as **1-D math**). Output is displayed using standard file formats supported by Jupyter such as LaTeX and PNG.



Jupyter Package

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

```
> 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.

```
> 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.

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

$$s := \begin{bmatrix} SMTLIB\ Session \\ 23844984 \\ Variables:\ 0 \\ Stack\ Depth:\ 0 \end{bmatrix}$$

```
> 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 );
```

$$M := \begin{bmatrix} DeepLearning\ Tensor \\ Shape:\ [5,\ 4,\ 3] \\ Data\ Type:\ float[8] \end{bmatrix}$$

You can use this to create slices of an existing Tensor.

```
> M2 := M[1..2,..,1..2];
```

$$M2 := \begin{bmatrix} \textit{DeepLearning Tensor} \\ \textit{Shape: [2, 4, 2]} \\ \textit{Data Type: float[8]} \end{bmatrix}$$

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.

```
> convert( M, Array );
```

$$\begin{bmatrix} 6.75313149056171 & 4.35116216422273 & 15.8836349575203 & 12.1491831021757 \\ 11.7160640627332 & 2.03390958544607 & 9.96030379277387 & 9.36755377418358 \\ 5.90730269483916 & 8.88558455121452 & 8.29299182444118 & 11.6766776384969 \\ 3.82283802557249 & 6.15935276225590 & 6.03589384550668 & 17.1526308248580 \\ 12.9928253114120 & 3.36829965586089 & 8.30668703969549 & 14.8079805405892 \end{bmatrix}$$

slice of 5 × 4 × 3 Array

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 := Constant([[3., 5.]], datatype=float[8]);
```

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

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

```
> tape := GradientTape();
```

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

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

```
> Enter( tape );
```

$$\begin{bmatrix} \textit{DeepLearning GradientTape} \\ \textit{<tensorflow.python.eager.backprop.GradientTape object at 0x0000022F309EC730>} \end{bmatrix}$$

```
> tape:-Watch(u);
```

```
> v := u^2;
```

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

```
> Exit( tape );
```

```
> grad := tape:-Gradient( v, u );
```

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

```
> convert( grad, Matrix );
```

$$\begin{bmatrix} 6. & 10. \end{bmatrix}$$

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] );
```

$$t1 := \begin{bmatrix} \textit{DeepLearning Tensor} \\ \textit{Shape: [3, 3]} \\ \textit{Data Type: float[8]} \end{bmatrix}$$

```
> 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 convert(..., DeepLearning) converts DeepLearning objects to their Python TensorFlow equivalents.

```
> pt2 := Python:-EvalString( "tf.random.uniform(shape=[3,4],
  maxval=20, dtype=tf.int32, seed=1701)" );
```

$$pt2 := \text{"<Python object: tf.Tensor(}$$
$$\text{[[9 2 7 13]}$$
$$\text{[3 18 12 11]}$$
$$\text{[9 19 12 12]], shape=(3, 4), dtype=int32)>"}$$

```
> t2 := convert( pt2, DeepLearning );
```

$$t2 := \begin{bmatrix} \textit{DeepLearning Tensor} \\ \textit{Shape: [3, 4]} \\ \textit{Data Type: integer[4]} \end{bmatrix}$$

```
> pt3 := Python:-EvalFunction( "tf.keras.layers.GaussianNoise", 5.5
  );
```

$$pt3 := \text{"<Python object: <tensorflow.python.keras.layers.noise.GaussianNoise object at 0x0000022F309D56A0>>"}$$

```
> t3 := convert( pt3, DeepLearning );
```

$$t3 := \begin{bmatrix} \textit{DeepLearning Layer} \\ \textit{<tensorflow.python.keras.layers.noise.GaussianNoise object at 0x0000022F309D56A0>} \end{bmatrix}$$

Additional Updates

Two-Dimensional Barcode Generation

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

```
> with(ImageTools):
```

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

$$qrCode := \begin{bmatrix} 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 1. & 0. & \dots \\ 0. & 1. & 1. & 1. & 1. & 1. & 0. & 1. & 1. & 0. & \dots \\ 0. & 1. & 0. & 0. & 0. & 1. & 0. & 1. & 1. & 0. & \dots \\ 0. & 1. & 0. & 0. & 0. & 1. & 0. & 1. & 1. & 0. & \dots \\ 0. & 1. & 0. & 0. & 0. & 1. & 0. & 1. & 0. & 0. & \dots \\ 0. & 1. & 1. & 1. & 1. & 1. & 0. & 1. & 1. & 0. & \dots \\ 0. & 0. & 0. & 0. & 0. & 0. & 0. & 1. & 0. & 1. & \dots \\ 1. & 1. & 1. & 1. & 1. & 1. & 1. & 1. & 0. & 1. & \dots \\ 1. & 1. & 0. & 0. & 1. & 1. & 0. & 0. & 0. & 1. & \dots \\ 1. & 0. & 1. & 0. & 0. & 0. & 1. & 0. & 0. & 0. & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \end{bmatrix}$$

25 × 25 Array

```
> ImageTools:-Embed( Scale( qrCode, 9, method=nearest ) );
```

Converting Data between Formats in Memory

- The convert command to objects of type Array, Matrix, Vector, string and ByteArray has been supplemented with an additional option, sourceformat. This reads encoded data from a string or ByteArray in a specified to the named type with no need for external files.

- Here we can read directly from a comma-delimited string to a Matrix:

```
> convert( "1,2\n3,4", Matrix, sourceformat="CSV" );
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

- 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.

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

$$T := \mathrm{table}\left(\left["\mathrm{lastname}" = "\mathrm{Raymond}", "\mathrm{firstname}" = "\mathrm{G}", "\mathrm{DOB}" = "1960\text{-}02\text{-}28"\right]\right)$$

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

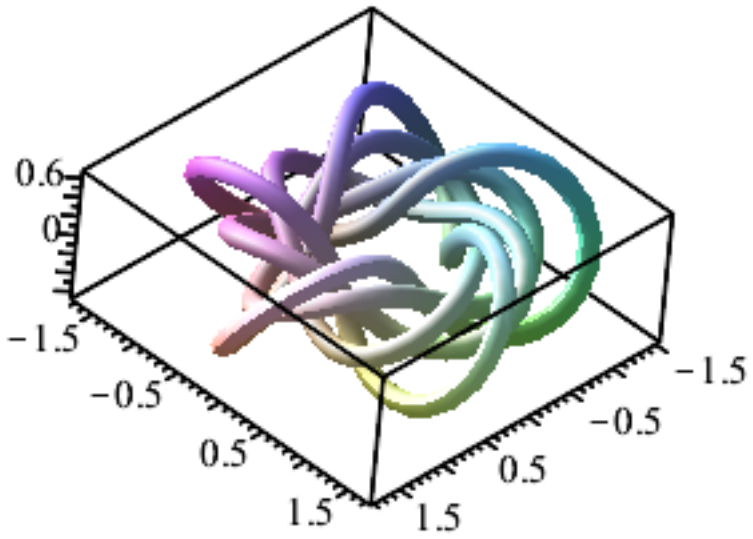
"{
"DOB": "1960-02-28",
"firstname": "G",
"lastname": "Raymond"
}"

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

"\left\{\frac{-b +\sqrt{-4 a c +b^{2}}}{2 a}, -\frac{b +\sqrt{-4 a c +b^{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" );
```

[71, 101, 110, 101, 114, 97, 116, 101, 100, 32, 98, 121, 32, 77, 97, 112, 108, 101, 32, 50, 48, 50, 50, 46, 48, 44, 32, 88, 56, 54, 32, 54, 52, 32, 76, 73, 78, 85, 88, 44, 32, 70, 101, 98, 32, 49, 54, 32, 50, 48, 50, 50, 44, 32, 66, 117, 105, 108, 100, 32, 73, 68, 32, 49, 53, 57, 52, 49, 48, 51, 32, 32, 32, 32, 32, 32, 32, 32, 32, 32, −96, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, −92, −34, −22, 63,…,
⋯ 238384 Array entries not shown]

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.