

In and Out

Edited by Paul Abbott

In and Out offers readers an opportunity to ask questions of the experts. *The Mathematica Journal* encourages readers to submit problems in care of the editor. Answers posted to the *Mathematica* news group, comp.soft-sys.math.mathematica, that appear here are edited for clarity and length.

■ In-flight Puzzle

Q: This puzzle appeared in an in-flight magazine: In the following diagram, place the numbers 1 through 9 so that each row, column, and 3×3 subsquare contains each number exactly once.

	3		9				8	
		6	2		3	7	9	
			1					
	2		3				7	
				7			6	4
1								
	5				4	9		
	7	2						
	9			5		8	3	

How can I solve this using *Mathematica*?

A: Bobby Treat (drbob@bigfoot.com) answers: Here is a naïve solution that depends on there being, at each stage, at least one cell that is fully determined and that does not check for contradictions. It works for the sort of problem you would expect to encounter in an in-flight magazine.

Input the 9×9 table with empty cells replaced by empty lists.

```

In[1]:= puzzle = 
$$\begin{pmatrix} \{\} & 3 & \{\} & 9 & \{\} & \{\} & \{\} & 8 & \{\} \\ \{\} & \{\} & 6 & 2 & \{\} & 3 & 7 & 9 & \{\} \\ \{\} & \{\} & \{\} & 1 & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 2 & \{\} & 3 & \{\} & \{\} & \{\} & 7 & \{\} \\ \{\} & \{\} & \{\} & \{\} & 7 & \{\} & \{\} & 6 & 4 \\ 1 & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 5 & \{\} & \{\} & \{\} & 4 & 9 & \{\} & \{\} \\ \{\} & 7 & 2 & \{\} & \{\} & \{\} & \{\} & \{\} & \{\} \\ \{\} & 9 & \{\} & \{\} & 5 & \{\} & 8 & 3 & \{\} \end{pmatrix};$$

```

PuzzleDisplay, written by David Park (djmp@earthlink.net), is used to display the puzzle.

```
In[2]:= Needs["Graphics`Colors`"]
```

```

In[3]:= PuzzleDisplay[matrix_] :=
Module[{workmatrix = Reverse[matrix], i, color, entry},
color[red[_]] := Red; color[_] := Black;
Show[Graphics[
{Navajo,
Rectangle[{0, 0}, {9, 9}],
SkyBlue,
MapThread[Rectangle[{#1, #2}, {#1 + 3, #2 + 3}] &,
{{0, 3, 3, 6}, {3, 0, 6, 3}}],
Black,
Table[Line[{{i, 0}, {i, 9}}], {i, 0, 9}],
Table[Line[{{0, i}, {9, i}}], {i, 0, 9}],
Table[{color[entry = Part[workmatrix, i, j]],
Text[entry /. {_} -> " ", red -> Identity], {j - 1/2, i - 1/2}],
{i, 1, 9}, {j, 1, 9}]
}],
AspectRatio -> Automatic,
TextStyle ->
{FontFamily -> "Helvetica", FontSize -> 9, FontWeight -> "Bold"},
ImageSize -> 150]];

```

```
In[4]:= PuzzleDisplay[puzzle]
```

From In[4]:=

	3		9				8	
		6	2		3	7	9	
			1					
	2		3				7	
				7			6	4
1								
	5				4	9		
	7	2						
	9			5		8	3	

The function **SubSquare** finds the 3×3 subtable corresponding to the element $\{i, j\}$.

```
In[5]:= SubSquare[p_List][i_, j_] :=
  p[[3 Floor[ $\frac{i-1}{3}$ ] + Range[3], 3 Floor[ $\frac{j-1}{3}$ ] + Range[3]]]
```

```
In[6]:= SubSquare[puzzle][4, 5]
```

```
Out[6]= {{3, {}, {}}, {{}, 7, {}}, {{}, {}, {}}}
```

Excluded determines what values *cannot* be entered in the cell $\{i, j\}$, for cells that are not yet known.

```
In[7]:= Excluded[p_List][i_, j_] := If[ListQ@p[[i, j]],
  Union@Flatten@{SubSquare[p][i, j], p[[All, j]], p[[i, All]]}, 0]
  SetOptions[Graphics, AspectRatio → Automatic];
```

Four values are excluded for the cell $\{4, 5\}$.

```
In[9]:= Excluded[puzzle][4, 5]
```

```
Out[9]= {2, 3, 5, 7}
```

OneChoice, applied to the cell $\{i, j\}$, returns the existing value (if any), which is the only choice if eight digits have been excluded, or $\{\}$ if the choice is not determined.

```
In[10]:= OneChoice[p_List][i_, j_] := Module[{e = Excluded[p][i, j]},
  Switch[Length@e,
    0, p[[i, j]],
    8, red@First@Complement[Range@9, e],
    _, {}]
  ]
```

There is only one choice for the cell $\{5, 2\}$.

```
In[11]:= OneChoice[puzzle][5, 2]
```

```
Out[11]= red[8]
```

Here is the iteration step, **Iterate**.

```
In[12]:= Iterate[p_, options__Rule] :=
  Module[{a = Array[OneChoice[p], {9, 9}]},
    If[Animate /. {options} /. Options[Iterate], PuzzleDisplay[a]];
    a /. red → Identity
  ];
  Options[Iterate] = {Animate → True};
```

And here is the animated solution. You can use the Up and Down Arrow keys to step through the animation once it has started.

```
In[14]:= solution = FixedPoint[Iterate, puzzle];
  SelectionMove[EvaluationNotebook[], All, GeneratedCell]
  FrontEndTokenExecute["OpenCloseGroup"]; Pause[0.5];
  FrontEndTokenExecute["SelectionAnimate"]
```

From In[14]:=

	3		9				8	
		6	2		3	7	9	
			1					
	2		3				7	
	8			7			6	4
1								
	5				4	9		
	7	2						
	9			5		8	3	

■ MonomialOrder

Q: An option for specifying the **MonomialOrder** in **GroebnerBasis** is to provide an explicit weight matrix. But I do not understand the syntax for doing this. Are there examples of this available? In addition, I would like to see a weight matrix for some of the standard monomial orderings.

A: Daniel Lichtblau (danl@wolfram.com) answers: What is required for n variables is an $n \times n$ rational (or integer) matrix of full rank that comprises a well-founded term ordering. This in turn is fulfilled iff the first nonzero entry in each column is positive. Here is an example from a set of problems posed as a challenge for the ISSAC 1997 conference.

```
In[1]:= polys = {8 w2 + 5 x w - 4 y w + 2 z w + 3 w + 5 x2 + 7 y2 + 7 z2 - 7 x + 2 x y -
7 y - 7 x z - 8 y z - 8 z + 8, 3 w2 - 5 x w - 3 y w - 6 z w + 9 w +
4 x2 + 9 y2 + 7 z2 + 7 x + 2 x y + 5 y - 2 x z + 6 y z + 7 z + 5,
-2 w2 + 9 x w + 9 y w - 7 z w - 4 w + 8 x2 + 6 y2 - 6 z2 + 8 x +
9 x y + 4 y - 3 x z - 7 y z + 8 z + 2, 7 w2 + 5 x w + 3 y w - 5 z w -
5 w + 2 x2 - 4 y2 - 4 z2 + 4 x + 9 x y + 6 y - 7 x z - 5 y z - 9 z + 2};
```

```
In[2]:= vars = {x, y, z, w};
```

For lexicographic ordering, you can use, for example, an identity matrix.

```
In[3]:= gbl1 = GroebnerBasis[polys, vars];
```

```
In[4]:= gbl2 = GroebnerBasis[polys, vars, MonomialOrder -> IdentityMatrix[4]];
```

```
In[5]:= gbl1 === gbl2
```

```
Out[5]= True
```

For degree reverse lexicographic ordering, you can use a matrix with first row set to all ones, next row all zeros except last entry negative one, then move the negative one step to the left in successive rows.

```

In[6]:= gbdr11 = GroebnerBasis[polys, vars,
      MonomialOrder → DegreeReverseLexicographic];
In[7]:= drlmat4 = {{1, 1, 1, 1}, {0, 0, 0, -1}, {0, 0, -1, 0}, {0, -1, 0, 0}};
In[8]:= gbdr12 = GroebnerBasis[polys, vars, MonomialOrder → drlmat4];
In[9]:= gbdr11 === gbdr12
Out[9]= True

```

■ Plot Zooming

Q: I know that you can change the **PlotRange** option to zoom into a plot region, but it is awkward for every zoom to use this option. Is there a way to zoom into a plot using the mouse?

A: Ingolf Dahl (ingolf.dahl@telia.com) answers: At library.wolfram.com/infocenter/MathSource/191 you will find a handy palette, written by Timothy Lottes, which is designed for this purpose. It works by controlling the **ImageRegion** parameter of the picture.

■ OpenCloseGroup

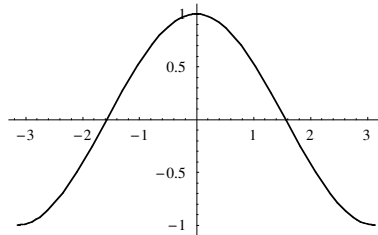
Q: I am using the following code to automatically close my graphics groups.

```

In[1]:= CloseAnimation[] :=
      (SelectionMove[EvaluationNotebook[], All, GeneratedCell];
      FrontEndTokenExecute["OpenCloseGroup"])
In[2]:= Table[Plot[Cos[x - t], {x, -π, π}, PlotRange → {-1.1, 1.1}], {t, 0, 6}];
      CloseAnimation[]

```

From In[2]:=



The problem is that when I only generate a single cell, **OpenCloseGroup** will generate a beep. How can I prevent that beep or test if **GeneratedCell** selected a single cell or several cells?

A: John Fultz (jfultz@wolfram.com) answers: You can test if multiple cells have been selected using

```
LinkWrite[$ParentLink, CellInformation[EvaluationNotebook[]]];
Length[LinkRead[$ParentLink]]
```

If the length is greater than 1, then you can use the **OpenCloseGroup** token.

```
In[3]:= CloseAnimation[] :=
  (SelectionMove[EvaluationNotebook[], All, GeneratedCell];
   LinkWrite[$ParentLink, CellInformation[EvaluationNotebook[]]];
   If[Length[LinkRead[$ParentLink]] > 1,
     FrontEndTokenExecute["OpenCloseGroup"]])
```

CellInformation is new to version 5 and is, in general, a very good way to test various properties of the selection without having to do a **NotebookRead** (which could get very expensive if you have a selection that consumes a lot of memory). **CellInformation** takes a notebook argument and returns a list of lists, one for each cell in the selection (only one if you have only selected a single cell or the content of a cell). Each list contains rules along with information about the object that is selected (similar to the way **NotebookInformation** works).

- The first argument of **CellInformation** must be a **NotebookObject**.
- The rules currently returned are as follows.

Style	the cell's style
ContentData	the type of cell (for example, GraphicsData , BoxData , TextData)
Evaluating	boolean, whether the cell is in the evaluation queue
Rendering	boolean, whether the cell is in the rendering queue

NeedsRendering	boolean, whether the cell would need to be rendered if displayed
CursorPosition	the position of the cursor; could be CellBracket or a list of numbers representing the content selection
FirstCellInGroup	boolean, whether the cell is a head cell of a group
CellSerialNumber	unique numerical ID for the cell
Formatted	boolean, whether the cell or the cell expression is being displayed
InlineCellPosition	if selection is in an inline cell, then the cell's position relative to the topmost cell

△ Of course, since this is part of the undocumented packet interface, we reserve the right to make changes to it. I hope to see this or something like it promoted to a documented function similar to **NotebookInformation** in the future.

■ String Trimming

Q: How can I trim a string, such as

```
In[1]:= mystring = " Have a great day. ";
```

in order to remove leading and trailing spaces?

A: Bobby Treat (drbob@bigfoot.com) writes: One way is to use **Characters** and **StringJoin**.

```
In[2]:= StringJoin[Characters[mystring] //.
           {" " .., x_> } -> {x}, {x_>, " " ..} -> {x}] // FullForm
```

```
Out[2]//FullForm= "Have a great day."
```

Brian Higgins (bghiggins@ucdavis.edu) writes: Another way to achieve your goal is to use the **trim** method in Java via *J/Link*.

```
In[3]:= Needs["JLink`"];
         InstallJava[];
```

```
In[5]:= astring = JavaNew["java.lang.String", mystring]
```

```
Out[5]= `JavaObject[java.lang.String]`
```

```
In[6]:= astring[trim[]] // FullForm
```

```
Out[6]//FullForm= "Have a great day."
```

At www.higgins.ucdavis.edu/Jlink.php there are various examples of integrating Java with *Mathematica*, including one that makes use of the Java regex (regular expressions) package that is useful for more complicated string manipulations.

In *Mathematica* 5.1, you also have the ability to work with regular expressions or use **StringReplace**. This trims white space off ends of strings.

```
StringReplace[mystring,
  (StartOfString~ ~Whitespace) | (Whitespace~ ~EndOfString) -> ""] //
InputForm
```

■ Root versus Radicals

Q: Version 5 returns the eigenvalues of the following simple matrix in terms of **Root** objects.

$$\text{In[1]:= mat} = \begin{pmatrix} -1 & 1 & 0 \\ 1 & -2 & 1 \\ 0 & 1 & -2 \end{pmatrix};$$

$\text{In[2]:= } \lambda = \text{Eigenvalues[mat]}$

$\text{Out[2]= } \{\text{Root}[#1^3 + 5 #1^2 + 6 #1 + 1 \&, 1],$
 $\text{Root}[#1^3 + 5 #1^2 + 6 #1 + 1 \&, 2], \text{Root}[#1^3 + 5 #1^2 + 6 #1 + 1 \&, 3]\}$

What does this output mean? I prefer the explicit answer that Version 4 returned. How do I coerce Version 5 to return an explicit answer?

A: Daniel Lichtblau (danl@wolfram.com) answers: The **Root** form is a concise way of expressing algebraic numbers via the minimal polynomial they satisfy, along with a canonical ordering in the complex plane (specified by the second argument of **Root**). To get a radical form, you can use **ToRadicals**.

$\text{In[3]:= rad} = \text{ToRadicals}[\lambda]$

$$\text{Out[3]= } \left\{ -\frac{5}{3} - \frac{\left(\frac{7}{2}\right)^{2/3} (1 + i\sqrt{3})}{3 \sqrt[3]{-1 + 3i\sqrt{3}}} - \frac{1}{6} (1 - i\sqrt{3}) \sqrt[3]{\frac{7}{2} (-1 + 3i\sqrt{3})}, \right.$$

$$\left. -\frac{5}{3} - \frac{\left(\frac{7}{2}\right)^{2/3} (1 - i\sqrt{3})}{3 \sqrt[3]{-1 + 3i\sqrt{3}}} - \frac{1}{6} (1 + i\sqrt{3}) \sqrt[3]{\frac{7}{2} (-1 + 3i\sqrt{3})}, \right.$$

$$\left. -\frac{5}{3} + \frac{7^{2/3}}{3 \sqrt[3]{\frac{1}{2} (-1 + 3i\sqrt{3})}} + \frac{1}{3} \sqrt[3]{\frac{7}{2} (-1 + 3i\sqrt{3})} \right\}$$

Clearly, this output is more complicated than the **Root** form.

```
In[4]:= {LeafCount[rad], LeafCount[λ]}
```

```
Out[4]= {206, 61}
```

In addition to size, there are other reasons to prefer the **Root** form:

- It is typically faster to obtain.
- It is numerically more stable to evaluate. In general, radical formulations are prone to numeric problems. **Root** objects do not have this liability.
- When the roots of an irreducible cubic are all real but not rational, the so-called “casus irreducibilis” shows that they still must be expressed in terms of i (mathworld.wolfram.com/CasusIrreducibilis.html). This means that numeric evaluation will give small imaginary parts unless, by happenstance, they exactly cancel. Small numeric error from round-off makes this unlikely.

```
In[5]:= N[λ]
```

```
Out[5]= {-3.24698, -1.55496, -0.198062}
```

```
In[6]:= N[rad]
```

```
Out[6]= {-3.24698 + 0. i, -1.55496 - 2.22045 × 10-16 i, -0.198062 + 1.11022 × 10-16 i}
```

- For sufficiently complicated algebraics, it is often faster to evaluate the **Root** form numerically, at least at high precision.

```
In[7]:= N[λ, 10000]; // Timing // First
```

```
Out[7]= 0.08 Second
```

```
In[8]:= N[rad, 10000]; // Timing // First
```

```
Out[8]= 0.35 Second
```

- Polynomial combinations of **Root** objects simplify using **RootReduce**.

```
In[9]:= Plus @@ λ // RootReduce
```

```
Out[9]= -5
```

```
In[10]:= Times @@ λ // RootReduce
```

```
Out[10]= -1
```

```
In[11]:= λ3 + 5 λ2 + 6 λ + 1 // RootReduce
```

```
Out[11]= {0, 0, 0}
```

- Derivatives of **Root** objects with respect to a parameter are expressed in terms of **Root** objects. This is useful for eigenvalue sensitivity analysis.

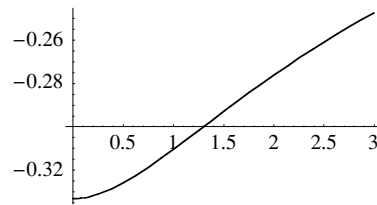
```
In[12]:= ∂x Root[#3 - x # + 1 &, 1]
```

```
Out[12]= 
$$\frac{\text{Root}[\#^3 - x \# + 1 \&, 1]}{3 \text{Root}[\#^3 - x \# + 1 \&, 1]^2 - x}$$

```

```
In[13]:= Plot[%, {x, 0, 3}]
```

From In[13]:=



So, one can avoid the **Root** form by using **ToRadicals**—but for all practical computation, you are better off working with **Root** objects.

■ Products of Polynomials

Q: I am computing large numbers of products of high-degree polynomials mod prime p , but the generic **PolynomialMod** function is too slow. How can I speed up this process?

A: Daniel Lichtblau (danl@wolfram.com) answers: There is fast dedicated technology built into the **Algebra`** context for manipulating dense univariate polynomials, represented by coefficient lists, modulo p .

```
In[1]:= ? Algebra`*
```

```
Algebra`
FactorSquareFreeModList
MatrixPowerMod
PartialSquareFreeDecompositionModList
PolynomialDerivativeModList
PolynomialDivisionModList
PolynomialExtendedGCDModList
PolynomialFactorModList
PolynomialGCDModList
PolynomialMakeMonicModList
PolynomialMinusModList
PolynomialPlusModList
PolynomialPowerMod
PolynomialPowerModList
PolynomialPthRootModList
PolynomialQuotientModList
PolynomialRemainderModList
PolynomialSubtractModList
PolynomialTimesModList
ReduciblePolynomialModListQ
SquareFreeDecompositionModList
SquareFreeModQ
SquareFreePartModList
```

The one you want is **PolynomialTimesModList**.

Start with code to generate random polynomials modulo some given p (which need not be prime).

```
In[2]:= randomPoly[deg_, mod_, x_] :=
      Table[Random[Integer, mod - 1], {deg + 1}].xRange[0, deg]
```

Construct a pair of degree 1200 random polynomials.

```
In[3]:= deg = 1200;
      SeedRandom[1234];
      p = Prime[1234];
      poly1 = randomPoly[deg, p, x];
      poly2 = randomPoly[deg, p, x];
```

Multiply them using **PolynomialMod**.

```
In[8]:= Timing[prod1 = PolynomialMod[poly1 poly2, p];] // First
```

Out[8]= 10.77 Second

Converting to coefficient lists, multiplying, and converting back is much faster.

```
In[9]:= Timing[
      listpoly1 = CoefficientList[poly1, x];
      listpoly2 = CoefficientList[poly2, x];
      prod2list =
      Algebra`PolynomialTimesModList[listpoly1, listpoly2, p];
      prod2 = prod2list.xRange[0, Length[prod2list]-1];] // First
```

Out[9]= 0.09 Second

Check the results for consistency.

```
In[10]:= prod1 === prod2
```

Out[10]= True

Generally speaking, you would want to do as much of the computation as possible using the list representation, and only convert from and to explicit polynomials, if at all, at the beginning and end of the process.

■ Closed Form Integral

Q: How can I compute a closed expression for

$$a_{k,l} = \int_0^1 P_k(x) P_l(2x - 1) dx,$$

where $P_n(z)$ is a Legendre polynomial?

A: Orthogonality of the Legendre polynomials requires that

$$P_k(x) = \sum_{l=0}^k (2l+1) a_{k,l} P_l(2x-1).$$

One can write $P_k(x)$ as a Gaussian hypergeometric function (functions.wolfram.com/Polynomials/LegendreP/26/01/01/0001),

$$P_k(x) = {}_2F_1\left(-k, k+1; 1; \frac{1-x}{2}\right).$$

Specializing equation (2.5) of Fields and Wimp [1], to the case where $p = 2$ and $q = 1$, one has

$${}_2F_1(a_1, a_2; b; zw) = {}_3F_2(\beta+1, a_1, a_2; \alpha+\beta+2, b; z) + \sum_{n=1}^{\infty} \frac{(a_1)_n (a_2)_n z^n}{(b)_n (n+\alpha+\beta+1)_n} \\ {}_3F_2(n+\beta+1, n+a_1, n+a_2; 2n+\alpha+\beta+2, b+n; z) P_n^{(\alpha,\beta)}(2w-1).$$

Put $a_1 = -k, a_2 = k+1, b = 1, z = 1/2, w = 1-x, \alpha = \beta = 0$ into this formula.

$$\text{In[1]:= } {}_2F_1(a_1, a_2; b; zw) = \sum_{n=0}^{\infty} \frac{(a_1)_n (a_2)_n z^n}{(b)_n (n+\alpha+\beta+1)_n} \\ {}_3F_2(n+\beta+1, n+a_1, n+a_2; 2n+\alpha+\beta+2, b+n; z) P_n^{(\alpha,\beta)}(2w-1) /. \\ \{a_1 \rightarrow -k, a_2 \rightarrow k+1, b \rightarrow 1, \alpha \rightarrow 0, \beta \rightarrow 0, z \rightarrow \frac{1}{2}, w \rightarrow 1-x, n \rightarrow l\}$$

$$\text{Out[1]:= } {}_2F_1\left(-k, k+1; 1; \frac{1-x}{2}\right) = \\ \sum_{l=0}^{\infty} \frac{2^{-k} {}_2F_1(l-k, -k+l+1; 2l+2; -1) P_l(2(1-x)-1) (-k)_l (k+1)_l}{(1)_l (l+1)_l}$$

The term $(-k)_l$ causes the summand to vanish if $l > k$, making the sum finite. We simplify the summand as follows.

In[2]:= % /. HoldPattern[Sum[s_, r_]] :> Sum[Simplify[s], r]

$$\text{Out[2]:= } {}_2F_1\left(-k, k+1; 1; \frac{1-x}{2}\right) = \\ \sum_{l=0}^{\infty} \frac{2^{-k} {}_2F_1(l-k, -k+l+1; 2l+2; -1) P_l(1-2x) (-k)_l (k+1)_l}{(1)_l (l+1)_l}$$

It is a good idea to verify that this expression is correct, at least for small k .

In[3]:= Table[%, {k, 0, 3}] // Simplify

Out[3]:= {True, True, True, True}

Finally, using the parity property, $P_l(2x - 1) = (-1)^l P_l(1 - 2x)$, we obtain the expansion coefficients $a_{k,l}$.

$$\begin{aligned} \text{In[4]:= } a_{k,l} &= \\ & \frac{(\text{Last}[\% \%] /. \text{HoldPattern}[\text{Sum}[s, _]] \text{:> Simplify}[s]) / ((2l + 1) P_l(1 - 2x))}{(-1)^l} \\ \text{Out[4]= } & \frac{(-1)^l 2^{-k} {}_2F_1(l - k, -k + l + 1; 2l + 2; -1) (-k)_l (k + 1)_l}{(2l + 1) (1)_l (l + 1)_l} \end{aligned}$$

We check this result for $0 \leq k, l \leq 3$.

$$\begin{aligned} \text{In[5]:= } & \text{Table}[\{l, k, \int_0^1 P_k(x) P_l(2x - 1) dx == a_{k,l}\}, \{l, 0, 3\}, \{k, 0, 3\}] \\ \text{Out[5]= } & \left(\begin{array}{cccc} \{0, 0, \text{True}\} & \{0, 1, \text{True}\} & \{0, 2, \text{True}\} & \{0, 3, \text{True}\} \\ \{1, 0, \text{True}\} & \{1, 1, \text{True}\} & \{1, 2, \text{True}\} & \{1, 3, \text{True}\} \\ \{2, 0, \text{True}\} & \{2, 1, \text{True}\} & \{2, 2, \text{True}\} & \{2, 3, \text{True}\} \\ \{3, 0, \text{True}\} & \{3, 1, \text{True}\} & \{3, 2, \text{True}\} & \{3, 3, \text{True}\} \end{array} \right) \end{aligned}$$

■ References

- [1] J. L. Fields and J. Wimp, "Expansions of Hypergeometric Functions in Hypergeometric Functions," *Mathematics of Computation*, **15**(76), 1961 pp. 390–395.

Paul Abbott

School of Physics, M013
The University of Western Australia
35 Stirling Highway
Crawley WA 6009, Australia
tmj@physics.uwa.edu.au
physics.uwa.edu.au/~paul