The Mathematica® Journal

Symbolic Solutions of Simultaneous First-Order PDEs in One Unknown

Célestin Wafo Soh

We propose and implement an algorithm for solving an overdetermined system of partial differential equations in one unknown. Our approach relies on the Bour–Mayer method to determine compatibility conditions via Jacobi–Mayer brackets. We solve compatible systems recursively by imitating what one would do with pen and paper: Solve one equation, substitute its solution into the remaining equations, and iterate the process until the equations of the system are exhausted. The method we employ for assessing the consistency of the underlying system differs from the traditional use of differential Gröbner bases, yet seems more efficient and straightforward to implement.

Introduction

The search of solutions of many problems leads to overdetermined systems of partial differential equations (PDEs). These problems comprise the computation of discrete symmetries of differential equations [1], the calculation of differential invariants [2] and the determination of generalized Casimir operators of a finite-dimensional Lie algebra [3]. In this article, we focus solely on the integration of simultaneous systems of scalar first-order PDEs; that is, our systems have at least two equations, one dependent variable (the unknown function) and several independent variables. Our ultimate goal is to automate the search of general symbolic solutions of these systems. The approach we adopt uses the Bour–Mayer method [4] to find compatibility conditions (i.e. obstructions to the integrability) of the underlying system of PDEs and to iteratively prepend these compatibility conditions to the system until a consistent or an inconsistent system is found. This differs from the traditional approach, which uses differential Gröbner bases [5] to discover compatibility conditions. When applicable, it has the advantage of being easy to implement and efficient. Recently, using machinery from differential geometry, Kruglikov and Lychagin [6] have extended the Bour–Mayer method to systems of PDEs in several dependent and independent variables of mixed orders (i.e. the orders of the individual equations in the system can be different). In our approach, for the situation where the completion process leads to a consistent system, we solve the latter by imitating what one would do with pen and paper: Solve one equation, substitute it into the next equation, and continue the process until the equations of the system are exhausted.

To fix ideas, consider a system of PDEs

$$F_i(x_1, x_2, \ldots, x_n, z, p_1, p_2, \ldots, p_n) = 0, \; i = 1, \ldots, m, \tag{1}$$

where x_1 to x_n are the independent variables, p_k is the partial derivative of the unknown function z with respect to x_k, and the rank of the Jacobian matrix $J = \left[\frac{\partial F_i}{\partial p_j}\right]$ is m. In the sequel, we will say that a property holds locally if it is true on an open ball of its domain of validity. The system of equations (1) is integrable (i.e. admits a locally smooth solution) provided the expressions p_1 to p_n derived from it locally satisfy the conditions

$$\frac{dp_i}{dx_j} = \frac{dp_j}{dx_i}, \; i < j = 1, \ldots, n. \tag{2}$$

To see this, consider a solution z of the system of equations (1). Then, locally, $dz = \sum_{i=1}^n p_i \, dx_i$. Thus, the latter differential form is locally exact. So, in particular, it is locally closed. Therefore, its exterior differential vanishes; that is, $d(\sum_{i=1}^n p_i \, dx_i) = 0$, or equivalently, after some calculations, $\sum_{i<j} \left(\frac{dp_i}{dx_j} - \frac{dp_j}{dx_i}\right) dx_j \wedge dx_i = 0$, which implies (2). Conversely, if the system of equations (2) is locally satisfied, then the differential form $\sum_{i=1}^n dx_i \, p_i$ is locally closed and by Poincaré's lemma, it is also locally exact. Hence, $dz = du(x_1, x_2, \ldots, x_n, z)$ for some locally smooth function u. Therefore z is locally defined by $z - u(x_1, x_2, \ldots, x_n, z) = c$, where c is an arbitrary constant.

Bour and Mayer (see e.g. [4]) showed that (1), subject to the condition on the Jacobian matrix of the F_i with respect to the p_j, is integrable if and only if the Jacobi–Mayer

$$[F_i, F_j] := \sum_{k=1}^n \frac{\partial F_i}{\partial x_k} \frac{\partial F_j}{\partial p_k} - \frac{\partial F_j}{\partial x_k} \frac{\partial F_i}{\partial p_k} = 0, i < j = 1, \ldots, m, \tag{3}$$

whenever (1) is satisfied. From now on, abbreviate the phrase "$[F_i, F_j] = 0$ whenever (1) is satisfied" to $[F_i, F_j]\,|_{(1)} = 0$.

For a given system of equations (1) satisfying the nondegeneracy condition mentioned, four cases arise.

The first case is when $m = n$ and all the Jacobi–Mayer brackets vanish whenever (1) is satisfied. In this case, we can solve (1) for p_1 to p_n. The solution of the system is then obtained by integrating the exact differential form $dz - \sum_{i=1}^n p_i \, dx_i$.

The second case is when there are distinct indices a and b such that $[F_a, F_b]\,|_{(1)} = \phi(x_1, x_2, \ldots, x_n, z) \neq 0$. Then (1) is incompatible and there are no solutions.

In the third case, $m < n$, and all the Jacobi–Mayer brackets vanish in (1). We must supplement (1) with additional equations until we get to the first or second case. These equations are obtained by solving the system of linear first-order PDEs $[F_\lambda, F_\mu] = 0$, where $\lambda = 1, \ldots, n$ and $\mu = m + 1, \ldots, n$. For example, we get the additional equation $F_{m+1} = a_1$, where a_1 is an arbitrary constant, by solving the system of linear first-order PDEs $[F_i, F_{m+1}] = 0$, where $i = 1, \ldots, m$. The solution of the completed system depends on $m - n + 1$ arbitrary constants. We obtain the general solution of the initial system of equations (1) by expressing one of the arbitrary constants as a function of the remaining ones, then eliminating the remaining constant between the resulting equations and their first-order partial derivatives with respect to the arbitrary constants.

In the fourth and final case, some brackets are zero in (1) and other brackets have the form $[F_a, F_b]\,|_{(1)} = \psi_{ab}(x_1, x_2, \ldots, x_n, z, p_1, p_2, \ldots, p_n)$, where the ψ_{ab} depend at least on some p_i. In this case, we must prepend the equations $\psi_{ab}(x_1, x_2, \ldots, x_n, z, p_1, p_2, \ldots, p_n) = 0$ to the equations in (1) and proceed as in the third case.

The procedure just described is the essence of the Bour–Mayer approach to the solution of (1). One has to solve overdetermined systems of linear scalar PDEs and ensure that the equations one adds to the initial system are compatible with them and that the equations of the resulting systems are linearly independent. In our implementation of the Bour–Mayer approach, we complete the initial system of equations (1) by prepending to it the appropriate compatibility constraints prescribed by Jacobi–Mayer brackets until we obtain either a compatible or an incompatible system. Starting from compatibility constraints, we iteratively solve the compatible system obtained by using the built-in function `DSolve`. The remainder of this article is devoted to the implementation and testing of this approach.

Implementation and Tests

Here we focus on the coding of the algorithm described in the introduction. Specifically, we start by iteratively solving a system of consistent first-order PDEs in one dependent variable. Then we implement the test of consistency of a system of first-order PDEs in one unknown. Finally, we couple the last two programs in such a way that a single function is used to compute the general solution of the input system when it exists or to indicate that it is inconsistent.

Iterative Solution of a Consistent System of First-Order PDEs in a Single Unknown

Our program for iteratively solving a compatible system of scalar first-order PDEs is made of the main function `solveCompatiblePDEs` and three helper functions, `PDEsToRules`, `rulesToSolution` and `functionFromExpression`.

`PDEsToRules` is a recursive function that takes as input the system `system` to be solved, the dependent variable `unknown`, the list of independent variables `xs`, a container `solutions` for the list of successive solutions, a list of equations `unsolvedEquations` that could not be solved, a string `symbol` that is used as a root to form the names of intermediate dependent variables, and a variable `count` that is used to count and name intermediate dependent variables. The output of `PDEsToRules` is a list of rules and a list of unsolved equations.

The function `PDEsToRules` mimics what one would do by hand when solving a system of first-order PDEs in one unknown: Solve an equation, substitute its solution into the remaining equations, and continue as long as possible. At each stage, the number of independent variables is reduced by one and it is necessary to rename the variables before proceeding. Also, the dependent variables are curried functions that must be undone to ensure that the chain rule is applied properly during substitution into the remaining PDEs. This is perhaps the trickiest part of our implementation.

The function `rulesToSolution` takes the output of `PDEsToRules` and converts it into the solution of the system to be solved. The helper function `functionFormExpression` converts an expression `expr` depending on several variables `vars` into a pure function of these variables. Finally, the function `solveCompatiblePDEs` composes `PDEsToRules` and `rulesToSolution` to solve a compatible system of scalar PDEs. Its inputs are like those of `PDEsToRules` and its output is formatted like that of `rulesToSolution`.

```
PDEsToRules[system_, unknown_, xs_, solutions_: {},
  unsolvedEquations_: {}, symbol_: "x", count_: 0] :=
 Module[{currentSol, newIndepVars = {}, newSyst,
   nameUnknown = "", temp, newUnsolvedEqs, f},
  If[system == {}, {solutions, unsolvedEquations},
   If[First@system === False, {{}, unsolvedEquations},
    If[Head[DSolveValue[First@system, unknown, xs]] ===
      DSolveValue, PDEsToRules[Rest[system], unknown,
      xs, solutions, Append[unsolvedEquations,
       First@system], symbol, count + 1],
     currentSol = DSolveValue[First@system, unknown,
       xs, GeneratedParameters → (C[# + count] &)];
     f = # /. {C[z_][t__][y__] :→ C[z] @@ {t, y}} &;
     FixedPoint[f, currentSol @@ xs] /.
      C[_][z__] :→ (newIndepVars = {z});
     currentSol @@ xs /. {C[z_] :→ ( nameUnknown = C[z])};
     currentSol = FixedPoint[f, currentSol];
     newSyst = Rest[system] /. {unknown → currentSol};
     newUnsolvedEqs = unsolvedEquations /.
       unknown → currentSol;
     newSyst = Select[newSyst, # =!= True &];
     If[newIndepVars ≠ {},
      temp = First@Solve[# == Unique[symbol] & /@ newIndepVars,
         xs];
      {newIndepVars, newSyst, newUnsolvedEqs} =
       Map[Simplify[# /. temp,
          TransformationFunctions →
           {Automatic, PowerExpand}] &,
```

```
        {newIndepVars, newSyst, newUnsolvedEqs}], ## & ];
     currentSol = Append[solutions, unknown → currentSol];
     PDEsToRules[newSyst, nameUnknown, newIndepVars,
      currentSol, newUnsolvedEqs, symbol, count + 1]]]]]
```

```
functionFromExpression[expr_, vars_] :=
 Function@expr /. MapIndexed[#1 → Slot @@ #2 &, vars]
```

```
rulesToSolution[list_, unknown_, xs_] :=
 Module[{s = First@list, temp1, temp2, count = 0},
  If[s === {}, {},
   temp1 = functionFromExpression[
     Fold[#1 /. #2 &, Values[First@s] @@ xs, Rest@s], xs]];
  temp2 = First@Rest@list /. unknown -> temp1;
  {unknown @@ xs -> temp1 @@ xs, temp2}]
```

```
solveCompatiblePDEs[system_, unknown_, xs_, solutions_: {},
  constraints_: {}, symbol_: "x"] :=
rulesToSolution[PDEsToRules[system, unknown, xs,
   solutions, constraints, symbol], unknown, xs]
```

▫ Compatibility Test and Completion

This subsection implements the compatibility test provided by the Bour–Mayer method as described in the introduction using `compatibilityQ`. The input to `compatibilityQ` is the underlying system of PDEs `system`, the dependent variable `ys` and the list of independent variables `xs`; `compatibilityQ` outputs a pair: the first element indicates whether the system is compatible and the second element gives the completed system.

The function `mayerBracketsSystem` computes the pairwise Jacobi–Mayer brackets of a system of PDEs according to equation (3) and in these brackets replaces some first-order partial derivatives of the unknown function obtained from the underlying system of PDEs. The function `derivativeQ` checks whether an expression contains a derivative of the unknown function.

```
mayerBrackets[f_, g_, ys_, xs_] :=
 Module[{p = D[ys @@ xs, #] & /@ xs,
   h = Function[{x, y}, D[x, #] & /@ y]},
  h[f, p].h[g, xs] - h[g, p].h[f, xs] // Simplify]
```

```
mayerBracketsSystem[system_, ys_, xs_] :=
 Module[{p = Solve[# == 0 & /@ system, D[ys @@ xs, #] & /@ xs]},
  If[p == {}, {},
   Flatten[
      Table[Table[mayerBrackets[system[[i]], system[[j]],
         ys, xs], {i, 1, j - 1}], {j, 1, Length[system]}]] /.
     First@p // Simplify]]

derivativeQ[expr_, ys_, xs_] :=
 Module[{temp = 0}, expr /. D[__][ys] @@ xs :> (temp = temp + 1;);
  temp > 0]

compatibilityQ[system_, ys_, xs_] :=
 Module[{brackets = mayerBracketsSystem[system, ys, xs],
   temp},
  If[brackets == {}, {False, {}},
   brackets = Select[brackets, Not[# === 0] &];
   If[brackets == {}, {True, system},
    temp = Select[brackets, ! derivativeQ[#, ys, xs] &];
    If[temp == {} && Length[system] + Length[brackets] <=
       Length[xs],
     compatibilityQ[Join[brackets, system], ys, xs],
     {False, {}}]]]]
```

Putting Everything Together

Here we use the functions defined so far to solve an overdetermined system of first-order PDEs in one unknown. The function `solveOverdeterminedScalarFirstOrderPDEs` takes as arguments the system to be solved, `system`, and its dependent and independent variables, `ys` and `xs`. The function `solutionQ` verifies whether a given rule `solutions` gives a solution of a system of first-order PDEs `system` in one unknown.

```
solveOverdeterminedScalarFirstOrderPDEs[system_, ys_, xs_] :=
  Module[
   {cSyst = compatibilityQ[system /. a_ == b_ :> a - b, ys, xs]},
   If[First@cSyst, solveCompatiblePDEs[
     # == 0 & /@ First@Rest@cSyst, ys, xs], {}]];

solutionQ[system_, ys_, xs_, solutions_] :=
  SelectFirst[system,
    Not[
      Simplify[
        # /. ys -> functionFromExpression[Values@solutions,
           xs]] === True] &] === Missing["NotFound"];
```

▫ Tests

This subsection is chiefly concerned with examples taken from various specified sources. For convenience, warnings are suppressed with the built-in function `Quiet`. Undefined global variables (`h`, `k`, `ht`, etc.) are used, so make sure there are no conflicts from your own session.

▪ *Test 1*

The examples presented here arise in the search of differential invariants of hyperbolic PDEs [2].

- **Example 1**

```
xs = {h, k, ht, hx, kt, kx};
```

```
ys = J @@ xs;
```

```
system1 = {
   h D[ys, ht] + k D[ys, kt] == 0,
   h D[ys, hx] + k D[ys, kx] == 0,
   h D[ys, h] + k D[ys, k] + 2 ht D[ys, ht] + hx D[ys, hx] +
     2 kt D[ys, kt] + kx D[ys, kx] == 0,
   h D[ys, h] + k D[ys, k] + ht D[ys, ht] + 2 hx D[ys, hx] +
     kt D[ys, kt] + 2 kx D[ys, kx] == 0
  };
```

```
solution1 =
 solveOverdeterminedScalarFirstOrderPDEs[system1, J, xs] //
  Quiet
```

$$\left\{J[h, k, ht, hx, kt, kx] \to C[4]\left[\frac{k}{h}, \frac{(-ht\,k + h\,kt)\,(-hx\,k + h\,kx)}{h^5}\right], \{\}\right\}$$

Except for example 9, `solutionQ` gives `True` for all systems, so it is only shown once here.

```
solutionQ[system1, J, xs, First@solution1]
```

```
True
```

- **Example 2**

```
With[
  {xs = {h, k, ht, hx, kt, kx, htt, htx, hxx, ktt, ktx, kxx}},
  ys = J @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    k D[ys, kxx] + h D[ys, hxx] == 0,
    k D[ys, ktt] + h D[ys, htt] == 0,
    k D[ys, kx] + h D[ys, hx] + 3 kx D[ys, kxx] +
      kt D[ys, ktx] + 3 hx D[ys, hxx] + ht D[ys, htx] == 0,
    k D[ys, kt] + h D[ys, ht] + kx D[ys, ktx] + 3 kt D[ys, ktt] +
      hx D[ys, htx] + 3 ht D[ys, htt] == 0,
    kxx D[ys, kxx] + 2 ktx D[ys, ktx] + 3 ktt D[ys, ktt] +
      hxx D[ys, hxx] + 2 htx D[ys, htx] + 3 htt D[ys, htt] +
      kx D[ys, kx] + 2 kt D[ys, kt] + hx D[ys, hx] +
      2 ht D[ys, ht] + k D[ys, k] + h D[ys, h] == 0,
    3 kxx D[ys, kxx] + 2 ktx D[ys, ktx] + ktt D[ys, ktt] +
      3 hxx D[ys, hxx] + 2 htx D[ys, htx] + htt D[ys, htt] +
      2 kx D[ys, kx] + kt D[ys, kt] + 2 hx D[ys, hx] +
      ht D[ys, ht] + k D[ys, k] + h D[ys, h] == 0
   },
   J, xs
  ]
 ] // Quiet
```

$$\left\{J[h,k,ht,hx,kt,kx,htt,htx,hxx,ktt,ktx,kxx]\to C[6]\left[\frac{k}{h},\ -\frac{-h\,htx+ht\,hx}{h^3},\ -\frac{hx\,kt-h\,ktx-\frac{ht\,(hx\,k-h\,kx)}{h}}{h^3},\ -\frac{h\left(-3\,hx^2\,k+3\,h\,hx\,kx-h\,(-hxx\,k+h\,kxx)\right)}{(hx\,k-h\,kx)^2},\ \frac{(ht\,k-h\,kt)\,(hx\,k-h\,kx)}{h^5},\ -\frac{\left(-3\,ht^2\,k+3\,h\,ht\,kt-h\,(-htt\,k+h\,ktt)\right)(hx\,k-h\,kx)^2}{h^9}\right],\ \{\}\right\}$$

- **Example 3**

```
With[
  {xs = {h, ht, hx, k, kt, kx, λ, κ}},
  ys = J @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    h D[ys, ht] + k D[ys, kt] == 0,
    h D[ys, hx] + k D[ys, kx] == 0,
    h D[ys, h] + k D[ys, k] + 2 ht D[ys, ht] + hx D[ys, hx] +
      2 kt D[ys, kt] + kx D[ys, kx] - λ D[ys, λ] == 0,
    h D[ys, h] + k D[ys, k] + ht D[ys, ht] + 2 hx D[ys, hx] +
      kt D[ys, kt] + 2 kx D[ys, kx] - κ D[ys, κ] == 0

   },
   J, xs
  ]
 ] // Quiet
```

$$\left\{J[h, ht, hx, k, kt, kx, \lambda, \kappa] \to C[4]\left[\frac{k}{h}, h\,\kappa\,\lambda, \frac{-ht\,k + h\,kt}{h^3\,\kappa}, \frac{(-hx\,k + h\,kx)\,\kappa}{h^2}\right], \{\}\right\}$$

Test 2

- **Example 4**

```
With[
  {xs = {t, x, y, z}},
  ys = f @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    -y D[ys, x] + z^2 D[ys, z] + 3 t z D[ys, t] - 4 z ys - 3 a t^2 ==
     0,
    -y D[ys, y] - z D[ys, z] - t D[ys, t] + ys == 0,
    -x D[ys, y] - D[ys, z] == 0
   },
   f, xs
  ]
 ] // Quiet
```

$$\left\{f[t, x, y, z] \to -\frac{3\sqrt{2}\,a\,t^2\,x}{\sqrt{2\,y - 2\,x\,z}\,\sqrt{y - x\,z}} + \frac{t^{3/2}\,C[4]}{\sqrt{2\,y - 2\,x\,z}}, \{\}\right\}$$

■ Test 3

Examples 5 and 6 come from Saltykow [7].

- **Example 5**

```
With[
  {xs = {y1, y2, y3, y4}},
  ys = f @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    D[ys, y1] - (1 / (y3 y4)) D[ys, y3] + (1 / y3^2) D[ys, y4] ==
     0,
    D[ys, y2] + (1 / y4) D[ys, y3] - (2 / y3) D[ys, y4] == 0
   },
   f, xs
  ]
 ] // Quiet
```

$\{f[y1, y2, y3, y4] \to C[2][y2 + y3\, y4, y1 + y3^2\, y4], \{\}\}$

- **Example 6**

```
With[
   {xs = {y1, y2, y3, y4}},
   ys = f @@ xs;
   solveOverdeterminedScalarFirstOrderPDEs[
    {
     2 y1 D[ys, y1] + 3 y2 D[ys, y2] + 4 y3 D[ys, y3] +
       5 y4 D[ys, y4] == 0,
     D[ys, y1] + 4 y1 D[ys, y3] + 5 y2 D[ys, y4] == 0,
     y2 D[ys, y3] + 2 (y3 - 2 y1^2) D[ys, y4] == 0
    },
    f, xs
   ]
  ] // Simplify // Quiet
```

$$\left\{f[y1, y2, y3, y4] \to C[3]\left[\frac{-4\, y1^4 - 5\, y1\, y2^2 + 4\, y1^2\, y3 - y3^2 + y2\, y4}{y1\, y2^2 \left(\frac{y2}{y1^{3/2}}\right)^{2/3}}\right], \{\}\right\}$$

■ *Test 4*

The two systems of PDEs treated here are in Mansion [4].

- **Example 7**

```
With[
  {xs = {y1, y2, y3, y4}},
  ys = v @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    2 y2 y4^2 D[ys, y1] + y3^2 y4 D[ys, y4] - y3^2 ys == 0,
    2 y2 D[ys, y2] - y4 D[ys, y4] - ys == 0,
    y2 y4^2 D[ys, y3] + y1 y3 y4 D[ys, y4] - y1 y3 ys == 0
   },
   v, xs
  ]
 ] // Quiet
```

$$\left\{v[y1,\ y2,\ y3,\ y4] \to \sqrt{2}\ \sqrt{y2}\ \sqrt{y2\ y4^2}\ C[3]\left[\frac{1}{2}\left(-y1\ y3^2 + y2\ y4^2\right)\right],\ \{\}\right\}$$

- **Example 8**

```
With[
   {xs = {y1, y2, y3, y4}},
   ys = z @@ xs;
   solveOverdeterminedScalarFirstOrderPDEs[
    {
     y3 D[ys, y1] + y4 D[ys, y2] - y1 D[ys, y3] - y2 D[ys, y4] ==
      0,
     y1 D[ys, y1] - y2 D[ys, y2] + y3 D[ys, y3] - y4 D[ys, y4] == 0
    },
    z, xs
   ]
  ] // Simplify // Quiet
```

$$\left\{z[y1,\ y2,\ y3,\ y4] \to C[2]\left[\frac{\sqrt{1+\frac{y1^2}{y3^2}}\ y3\ (y1\ y2 + y3\ y4)}{\sqrt{2}\ \sqrt{y1^2+y3^2}},\ \frac{\sqrt{1+\frac{y1^2}{y3^2}}\ y3\ (y2\ y3 - y1\ y4)}{\sqrt{2}\ \sqrt{y1^2+y3^2}}\right],\ \{\}\right\}$$

- **Example 9**

```
sol9 = With[
    {xs = {y1, y2, y3, y4, y5}},
    ys = f @@ xs;
    solveOverdeterminedScalarFirstOrderPDEs[
     {
      D[ys, y1] D[ys, y5] - y2 y4 == 0,
      D[ys, y2] D[ys, y4] - y1 y5 == 0
     },
     f, xs
    ]
   ] // Simplify // Quiet
```

$$\Big\{f[y1, y2, y3, y4, y5] \to C[1][y3, y4, y5] + \frac{y1\, y4\, y5}{C[2][y3, y4, y5]} + y2\, C[2][y3, y4, y5],$$

$$\Big\{\Big(x127\, x128 \Big(\frac{x127\, y1}{C[2][x126, x127, x128]} + C[1]^{(0,0,1)}[x126, x127, x128] + y2\, C[2]^{(0,0,1)}[x126, x127, x128] - \frac{x127\, x128\, y1\, C[2]^{(0,0,1)}[x126, x127, x128]}{C[2][x126, x127, x128]^2}\Big)\Big) \Big/ C[2][x126, x127, x128] == x127\, y2,$$

$$\frac{x127\, x128\, y1\, C[2]^{(0,1,0)}[x126, x127, x128]}{C[2][x126, x127, x128]} == C[2][x126, x127, x128] \Big(C[1]^{(0,1,0)}[x126, x127, x128] + y2\, C[2]^{(0,1,0)}[x126, x127, x128]\Big)\Big\}\Big\}$$

The second entry of `sol9` shows that there are two PDEs that were not solved. It is straightforward to separate these PDEs with respect to `y1` and `y2` to obtain new PDEs that are easily solved using the built-in function `DSolve`. The separation can be done automatically through the following one-liner.

```
# == 0 & /@
  Flatten[
   Values /@ Map[CoefficientRules[#, {y1, y2}] &,
     # /. (a_ == b_) :> a - b & /@ sol9[[2]]]] // Simplify
```

$$\left\{x127\,x128\left(-1+\frac{x128\,C[2]^{(0,0,1)}[x126,x127,x128]}{C[2][x126,x127,x128]}\right)==0,\right.$$
$$x127\left(-1+\frac{x128\,C[2]^{(0,0,1)}[x126,x127,x128]}{C[2][x126,x127,x128]}\right)==0,$$
$$\frac{x127\,x128\,C[1]^{(0,0,1)}[x126,x127,x128]}{C[2][x126,x127,x128]}==0,$$
$$\frac{x127\,x128\,C[2]^{(0,1,0)}[x126,x127,x128]}{C[2][x126,x127,x128]}==0,$$
$$C[2][x126,x127,x128]\,C[2]^{(0,1,0)}[x126,x127,x128]==0,$$
$$\left.C[2][x126,x127,x128]\,C[1]^{(0,1,0)}[x126,x127,x128]==0\right\}$$

■ *Test 5*

- **Example 10**

```
With[
  {xs = {x, y, z, t}},
  ys = f @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    D[ys, t] + (1 - x) D[ys, x] / t == 0,
    D[ys, z] + (y - (x - 1) t) D[ys, x] / (z t) == 0,
    D[ys, y] + D[ys, x] / t == 0
   },
   f, xs
  ]
 ] // Quiet
```

```
{f[x, y, z, t] → C[3][(-t (-1 + x) + y) z], {}}
```

- **Example 11**

```
With[
  {xs = {x, y, z}},
  ys = u @@ xs;
  solveOverdeterminedScalarFirstOrderPDEs[
   {
    D[ys, x] == 4 Sin[y] Sin[y] Cos[z],
    1 / x D[ys, y] == 4 Cos[z] Sin[2 y],
    1 / (x Sin[y]) D[ys, z] == -4 Sin[y] Sin[z]
   },
   u, xs
  ]
 ] // Quiet
```

```
{u[x, y, z] → C[3] - x Cos[2 y - z] + 2 x Cos[z] - x Cos[2 y + z], {}}
```

- **Example 12**

```
With[
  {xs = {x, y}},
  ys = z @@ xs;
  solveCompatiblePDEs[
   {
    D[ys, x] == a E^(y - ys),
    D[ys, y] == b E^(y - ys) + 1
   },
   z, xs
  ]
 ] // Quiet
```

```
{z[x, y] → Log[a ⅇ^y x + b ⅇ^y y + ⅇ^y C[2]], {}}
```

The last example is due to Boole [8].

- **Example 13**

```
With[
   {xs = {x, y, z, t}},
   ys = p @@ xs;
   solveOverdeterminedScalarFirstOrderPDEs[
    {
     D[ys, x] + (t + x y + x z) D[ys, z] + (y + z - 3 x) D[ys, t] == 0,
     D[ys, y] + (x z t + y - x y) D[ys, z] + (z t - y) D[ys, t] == 0
    },
    p, xs
   ]
  ] // Simplify // Quiet
```

$$\left\{p[x, y, z, t] \to C[3]\left[-t\,x - x^3 - \frac{y^2}{2} + z\right], \{\}\right\}$$

Conclusion

This article has introduced and implemented an algorithm based on the Bour–Mayer method for solving an overdetermined system of PDEs in one unknown. We have demonstrated the efficiency of our approach through the consideration of 13 examples.

Acknowledgments

I gratefully acknowledge partial financial support from the DST-NRF Centre of Excellence in Mathematical and Statistical Sciences, School of Computer Science and Applied Mathematics, University of the Witwatersrand, Johannesburg, South Africa. I thank Prof. F. M. Mahomed for securing the necessary funds and his team for the hospitality during my visit last summer. This article is dedicated to my daughter Katlego on her sixteenth birthday.

References

[1] P. E. Hydon, “How to Construct the Discrete Symmetries of Partial Differential Equations,” *European Journal of Applied Mathematics*, **11**(5), 2000 pp. 515–527.

[2] I. K. Johnpillai, F. M. Mahomed and C. Wafo Soh, “Basis of Joint Invariants for $(1+1)$ Linear Hyperbolic Equations,” *Journal of Nonlinear Mathematical Physics*, **9**(Supplement 2), 2002 pp. 49–59. doi:10.2991/jnmp.2002.9.s2.5.

[3] J. C. Ndogmo and P. Winternitz, “Generalized Casimir Operators of Solvable Lie Algebras with Abelian Nilradicals,” *Journal of Physics A: Mathematical and General*, **27**(8), 1994 pp. 2787–2800. iopscience.iop.org/article/10.1088/0305-4470/27/8/016/meta.

[4] P. Mansion, *Théorie des équations aux dérivées partielles du premier ordre*, Paris: Gauthier-Villars, 1875.

[5] B. Buchberger and F. Winkler, *Gröbner Bases and Applications*, Cambridge: Cambridge University Press, 1998.

[6] B. Kruglikov and V. Lychagin, "Compatibility, Multi-brackets and Integrability of Systems of PDEs," *Acta Applicandæ Mathematicæ*, **109**(1), 2010 pp. 151–196. doi:10.1007/s10440-009-9446-0.

[7] N. Saltykow, "Méthodes classiques d'intégration des équations aux dérivées partielles du premier ordre à une fonction inconnu," *Mémorial des sciences mathématiques*, **50**, 1931 pp. 1–72. www.numdam.org/item?id=MSM_ 1931__ 50__ 1_ 0.

[8] G. Boole, "On Simultaneous Differential Equations of the First Order in Which the Number of the Variables Exceeds by More Than One the Number of the Equations," *Philosophical Transactions of the Royal Society of London*, **152**(5), 1862 pp. 437–454. doi:10.1098/rstl.1862.0023.

C. W. Soh, "Symbolic Solutions of Simultaneous First-Order PDEs in One Unknown," *The Mathematica Journal*, 2018. dx.doi.org/doi:10.3888/tmj.20-2.

About the Author

Dr. C. Wafo Soh is currently an associate professor of mathematics at Jackson State University and a visiting associate professor of applied mathematics at the University of the Witwatersrand. He is the cofounder of the South African startup Recursive Thinking Consulting, which specializes in process mining.

Célestin Wafo Soh [1,2]
[1] *Department of Mathematics and Statistical Science*
JSU Box 1760, Jackson State University
1400 JR Lynch Street
Jackson, MS 39217
Celestin.Wafo_Soh@jsums.edu

[2] *DST-NRF Centre of Excellence in Mathematical and Statistical Sciences*
School of Computer Science and Applied Mathematics, University of the Witwatersrand
Johannesburg, Wits 2050, South Africa