The Brachistochrone Differential Equation

François Boulier*

May 14, 2023

Contents

This text corresponds to a talk I have given at CRIStAL / CFHP working group seminar on February 15, 2023. I had been looking for a good introductory example for the CIMPA School *Algebraic and Tropical Methods for Solving Differential Equations*. It shows that polynomial ODE do naturally arise. It relates the existence problem of formal power series with the numerical integration of initial value problems. It introduces Puiseux series. Computations are performed using the `Python/sympy` version of the `DifferentialAlgebra` package [1].

1 Derivation of the Differential Equation

1.1 Formulation of the Problem

A point with mass m is forced to follow a curve $y(x)$ in the (x, y)-plane between two fixed points $(x, y) = (a, y_a) = (0, 0)$ and (b, y_b). Its movement follows the gravitational law. Its

*Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille, F-59000 Lille, France.

initial speed is zero. There is no friction. The problem is: find the curve $y(x)$ which minimizes the time needed to reach the point (b, y_b).

Let $s(x)$ be the function which gives the arc length of the curve $y(x)$. The following formula is classical:

$$s(x) \;=\; \int_a^x \sqrt{1+\dot{y}^2(\xi)}\,\mathrm{d}\xi\,.$$

Differentiating w.r.t. x we get

$$\frac{\mathrm{d}s}{\mathrm{d}x}(x) \;=\; \sqrt{1+\dot{y}^2(x)}\,. \tag{1}$$

Now, viewing the time t as a function of x, by the chain rule we have

$$\frac{\mathrm{d}s}{\mathrm{d}x} \;=\; \frac{\mathrm{d}s}{\mathrm{d}t}\,\frac{\mathrm{d}t}{\mathrm{d}x}. \tag{2}$$

Since the motion is frictionless, at any x, the kinetic energy is equal to the gravitational potential energy. Since $y_a = 0$ and assuming the y axis to be oriented downward:

$$\frac{1}{2}\,m\,\left(\frac{\mathrm{d}s}{\mathrm{d}t}\right)^2 \;=\; m\,g\,y\,. \tag{3}$$

Combining (1), (2) and (3) we get

$$\frac{\mathrm{d}t}{\mathrm{d}x} \;=\; \sqrt{\frac{1+\dot{y}^2}{2\,g\,y}}.$$

Denote $t(x)$ the time at which abscissa x is reached. The problem consists in minimizing

$$t(b)-t(a) \;=\; \int_a^b \sqrt{\frac{1+\dot{y}^2}{2\,g\,y}}\,\mathrm{d}x$$

We are thus looking to the function $y(x)$ which minimizes the functional

$$y \;\mapsto\; \int_a^b \sqrt{\frac{1+\dot{y}^2}{2\,g\,y}}\,\mathrm{d}x\,.$$

Introduce the following Lagrangian:

$$\mathscr{L}(y,\dot{y}) \;=\; \sqrt{\frac{1+\dot{y}^2}{2\,g\,y}}.$$

The Beltrami formula (a special case of the Euler-Lagrange formula) applies[1] and the optimal $y(x)$ function satisfies the following (necessary) equation where c is a constant:

$$\mathscr{L} - \dot{y}\,\frac{\partial \mathscr{L}}{\partial \dot{y}} \;=\; c\,.$$

[1] The assumption that the starting and the ending point are fixed is used.

1.2 The Differential System to be Solved

The equation for $y(x)$ can be obtained by differential elimination, the following computations show. The next commands load `sympy` and `DifferentialAlgebra` and allow some ASCII art pretty-printing.

[1]:
```
from sympy import *
from DifferentialAlgebra import *
init_printing ()
```

The next commands define the independent variable `x`, three constants and three differential indeterminates.

[2]:
```
x,g,c,D = var('x,g,c,D')
L,S_L,y = function('L,S_L,y')
```

The next commands define the differential polynomial ring `R`, endowed with a ranking. The ranking is defined by the `blocks` list. The differential indeterminate y and the three constants appear on the rightmost end of the `blocks` list: this aims at telling the Rosenfeld-Gröbner algorithm to seek an equation which depends on these differential indeterminates and their derivatives only.

[3]:
```
params = [g,c,D]
R = DifferentialRing (derivations=[x], blocks=[L,S_L,y,params],␣
 ↪parameters=params)
R
```

We can now write down the system to be solved. The symbols `L` and `S_L` correspond to the Lagrangian $\mathscr{L}$ and its "separant" $\partial\mathscr{L}/\dot{y}$. In differential algebra, all equations are differential polynomials. In order to handle the square root, the trick consists in defining `L` and `S_L` through implicit equations. The last equation only aims at renaming a constant as D (it is denoted L in [2, section I.5, page 23]).

[4]:
```
syst = [ Eq(L(x)**2, (1+Derivative(y(x),x)**2)/(2*g*y(x))),
         Eq(2*L(x)*S_L(x), Derivative(y(x),x)/(g*y(x))),
         Eq(L(x) - Derivative(y(x),x)*S_L(x), c),
         Eq(D, 1/(2*g*c**2))]
syst
```

[4]:
$$\left[L^2(x) = \frac{\left(\frac{d}{dx}y(x)\right)^2 + 1}{2gy(x)},\ 2L(x)\,\mathrm{S_L}\,(x) = \frac{\frac{d}{dx}y(x)}{gy(x)},\ L(x) - \mathrm{S_L}\,(x)\frac{d}{dx}y(x) = c,\ D = \frac{1}{2c^2g}\right]$$

1.3 The Differential Ideal Defined by the System

Differential elimination is performed over `syst`. Computations are led by the ranking. The result is a list of two regular differential chains / characteristic sets:

[5]:
```
ideal = R.RosenfeldGroebner (syst)
ideal
```

Here are the differential polynomials which constitute the two regular differential chains. For a better understanding, they are displayed in "solved form" i.e. as equations with their leading ranks on the lefthand sides and their tails divided by their initials on the righthand sides:

[6]:
```
[ C.equations(solved=True) for C in ideal ]
```

[6]:
$$\left[\left[g = \frac{1}{2Dc^2},\ \left(\frac{d}{dx}y(x)\right)^2 = \frac{D - y(x)}{y(x)},\ \mathrm{S_L}\,(x) = c\frac{d}{dx}y(x),\ L(x) = \frac{Dc}{y(x)}\right],\right.$$
$$\left.\left[g = \frac{1}{2Dc^2},\ y(x) = D,\ \mathrm{S_L}\,(x) = 0,\ L(x) = c\right]\right]$$

1.4 Extraction of the Brachistochrone Equation

The sought ODE is the second equation of the first chain. It only depends on $y(x)$, its first derivative and the constant D. It was exhibited by the differential elimination process because of the chosen ranking. The next commands show how to extract it.

[7]:
```
leader = var('leader')
derivative = function('derivative')
C = ideal[0]
edo = C.equations(selection=Eq(leader,derivative(y(x))))[0]
edo
```

[7]:
$$-D + y(x)\left(\frac{d}{dx}y(x)\right)^2 + y(x)$$

The brachistochrone equation can be written as

$$\dot{y}^2 \;=\; \frac{D}{y} - 1 \quad \text{or} \quad y\,\dot{y}^2 + y \;=\; D\,. \tag{4}$$

The solution is known to be a cycloid generated by a circle of diameter D.

1.5 A General and a Singular Solution

Each regular differential chain defines some differential ideal. The *intersection* of these two differential ideals is the radical of the differential ideal generated by `syst`. Therefore, the solution set of `syst` is the *union* of the solution sets of the regular differential chains.

The first regular differential chain defines the *general solution* of `syst`, which is a family of curves satisfying an implicit order one ODE (the brachistochrone equation). The second regular differential chain defines a degenerate solution: the constant function $y(x) = D$. Over some examples, it happens that degenerate solutions are *particular cases* of the general

solution (obtained from the general solution, for particular initial values). Over this example, the degenerate solution is a genuine *singular solution*.

Let us prove this claim. Denote $\mathfrak{S} = \{\texttt{syst}\}$ the radical of the differential ideal generated by `syst`. Denote A_1, A_1 the two regular differential chains and $\mathfrak{A}_i = [A_i] : {H_{A_i}}^\infty$ the differential ideals that they define, for $i = 1, 2$. We have $\mathfrak{S} = \mathfrak{A}_1 \cap \mathfrak{A}_2$. In order to prove our claim, let us admit it is sufficient to prove that $\mathfrak{A}_1$ and $\mathfrak{A}_2$ are disjoint. For this, it is sufficient to exhibit some product $p_1\, p_2$ of two differential polynomials such that $p_1\, p_2 \in \mathfrak{S}$ and $p_i \in \mathfrak{A}_i$ but $p_i \notin \mathfrak{A}_j$ for $i = 1, 2$ and $j \neq i$.

Start with the brachistochrone equation. It necessarily belongs to $\mathfrak{A}_1$. To make sure it belongs to $\mathfrak{S}$ it is sufficient to check that its *normal form* with respect to each regular differential chain is zero:

[8]:
```
[ C.normal_form(edo) for C in ideal ]
```

[8]:
$$[0,\ 0]$$

It turns out that its derivative factors:

[9]:
```
dedo = R.differentiate (edo, x).factor ()
dedo
```

[9]:
$$\left(2y(x)\frac{d^2}{dx^2}y(x) + \left(\frac{d}{dx}y(x)\right)^2 + 1\right)\frac{d}{dx}y(x)$$

Let us extract the two factors:

[10]:
```
p2 = Derivative(y(x),x)
p2
```

[10]:
$$\frac{d}{dx}y(x)$$

[11]:
```
p1 = dedo / p2
p1
```

[11]:
$$2y(x)\frac{d^2}{dx^2}y(x) + \left(\frac{d}{dx}y(x)\right)^2 + 1$$

The following normal forms computations conclude the proof of our claim. They also prove that $\mathfrak{S}$ is not a *prime* differential ideal.

[12]:
```
[ C.normal_form (p1) for C in ideal ], [ C.normal_form (p2) for C in ideal ]
```

[12]:
$$\left([0,\ 1]\,,\ \left[\frac{d}{dx}y(x),\ 0\right]\right)$$

Remark that, in general, no algorithm is known to decide the inclusion between two differential ideals defined by regular differential chains (it is a famous open problem in differential

algebra). This is due to the fact that regular differential chains are *not* generating families of the differential ideals they define.

2 Numerical Integration of the Brachistochrone Equation

2.1 The Straightforward Approach Fails

Let us assume that $a = 0$ for simplicity (from now on, a is denoted x_0). In the former section, we assumed $y(a) = y(x_0) = y_0 = 0$. First transform ODE (4) into explicit form:

$$\begin{array}{rcl} \dot{y} & = & f(x,y)\,, \\ y(x_0) & = & y_0\,, \end{array} \tag{5}$$

where (it is classical to view f as a function of x and y though the ODE is here *autonomous*):

$$f(x,y) \;=\; \sqrt{\frac{D}{y}-1}\,.$$

Let us consider the Runge mid-point formula, which is an order 2 explicit Runge-Kutta method, for this initial value problem:

$$\begin{array}{rcl} k_1 & = & f(x_0,y_0)\,, \\ k_2 & = & f(x_0+\frac{1}{2}\,h,\; y_0+\frac{1}{2}\,h\,k_1)\,, \\ y_1 & = & y_0+h\,k_2\,. \end{array} \tag{6}$$

In the above scheme, h is the step size, (x_0, y_0) is the starting point. The next point is supposed to be $(x_0 + h, y_1)$. Unfortunately, for $(x_0, y_0) = (0, 0)$, the function call $f(x_0, y_0)$ raises a division-by-zero error. Let us recall [2, section I.7, Thm 7.4] which gives a *sufficient* condition for the existence and uniqueness of a solution.

Theorem 1 *Assume U to be an open set in $\mathbb{R}^2$ and let f and $\partial f/\partial y$ be continuous on U. Then, for every $(x_0, y_0) \in U$, there exists a unique solution of (5), which can be continued up to the boundary of U (in both directions).*

In our case,

$$\frac{\partial f}{\partial y} \;=\; -\frac{D}{2\,y^2\,\sqrt{\frac{D}{y}-1}} \tag{7}$$

becomes infinite at (x_0, y_0) hence [an open set U which contains it] does not satisfy the hypotheses of Theorem 1.

2.2 Relationship with the Existence of a Formal Power Series Solution

The numerical integration problem is also related to the fact ODE (4) has no power series solution for the initial value $y_0 = 0$. Indeed, explicit Runge-Kutta methods are derived as

follows [2, section II.1, page 132]. First compute the Taylor expansion of the exact solution of (5). It is a series in h:

$$y(x_0+h) \;=\; y_0 + h\,f(x_0,y_0) + \frac{h^2}{2}\,((f_x + f_y\,f)(x_0,y_0)) + \cdots$$

As an example, let us take Runge mid-point formula and replace, for a better understanding, the numerical coefficients by parameters:

$$\begin{aligned} k_1 &= f(x_0,y_0)\,,\\ k_2 &= f(x_0 + c_2\,h,\; y_0 + a_{21}\,h\,k_1)\,,\\ y_1 &= y_0 + h\,(b_1\,k_1 + b_2\,k_2)\,. \end{aligned}$$

Then compute the Taylor expansion of y_1, viewed as a function of h,

$$y_1(h) \;=\; y_0 + h\,(b_1+b_2)\,f(x_0,y_0) + h^2\,b_2\,((c_2\,f_x + a_{21}\,f_y\,f)(x_0,y_0)) + \cdots$$

and require that both series coincide termwise (for any function f hence for any ODE) up to some order. One solution is $(c_2, a_{21}, b_1, b_2) = (\frac{1}{2}, \frac{1}{2}, 0, 1)$ which gives Runge mid-point formula. Some further analysis permits to determine the *order* of the Runge-Kutta scheme, which measures its error, as a function of h.

Thus, if (5) has no formal power series solution or non unique formal power series solutions, at (x_0, y_0) then the above derivation does not make sense and Runge-Kutta methods are not supposed to apply.

A remark on differential elimination. In principle, the above computation of the conditions that the Runge-Kutta coefficients must satisfy is an elimination problem. However, it is not easy to carry out in the framework of differential algebra because computations involve applications of the chain rule and evaluations at $x = x_0$. These operations are not handled by classical differential algebra, though some attempts to handle the chain rule have been undertaken in [3].

2.3 The True Solution

Observe that the above problem is due to the ODE since the sought cycloid (Figure 1) is the solution of the following system which can be evaluated at $\theta = 0$:

$$\begin{aligned} x(\theta) &= \frac{D}{2}\,(\theta - \sin\theta)\,,\\ y(\theta) &= \frac{D}{2}\,(1 - \cos\theta)\,. \end{aligned} \tag{8}$$

2.4 Existence of a Puiseux Series Solution

Fortunately [2, section I.5, page 23], ODE (4) has a Puiseux series solution. In order to obtain it, we need the order zero term D to be cancelled by $y\,\dot{y}^2$ evaluated at a monomial in x^q. We thus need $q = \frac{2}{3}$. Let us compute the three first terms of the Puiseux series solution. First define a series `solp` in $x^{\frac{2}{3}}$ with parametric coefficients:

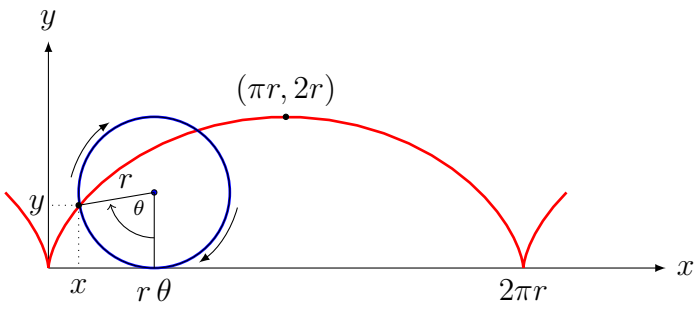

Figure 1: The picture is borrowed from `https://tex.stackexchange.com/questions/196957`. The radius r is $D/2$ with our notations.

[13]:
```
a0,a1,a2 = var('a0,a1,a2')
solp = a0*x**(2/Integer(3)) + a1*x**(4/Integer(3)) + a2*x**(6/Integer(3))
solp
```

[13]: $a_0 x^{\frac{2}{3}} + a_1 x^{\frac{4}{3}} + a_2 x^2$

[14]:
```
values = {}
values[a1] = -9/(20*a0)
values[a2] = -243/(2800*a0**3)
values[a0] = (9*D/4)**(1/Integer(3))
y_at_solp = R.evaluate (edo, {y(x):solp})
```

Then evaluate the ODE (4) at this series and figure out values annihilating the resulting series up to x^2:

[15]:
```
y_at_solp.subs(values).subs(values).doit().expand()
```

[15]: $$\frac{207x^2}{3500D} - \frac{19683x^4}{85750000D^3} - \frac{1809\sqrt[3]{2}\cdot 3^{\frac{2}{3}}x^{\frac{8}{3}}}{245000D^{\frac{5}{3}}} - \frac{729\cdot 2^{\frac{2}{3}}\sqrt[3]{3}x^{\frac{10}{3}}}{350000D^{\frac{7}{3}}}$$

The truncated Puiseux series solution is obtained by substituting the figured out values in the parametric series `solp`:

[16]:
```
soln = R.evaluate (R.evaluate (solp, values), values)
soln
```

[16]: $$\frac{\sqrt[3]{2}\cdot 3^{\frac{2}{3}}\sqrt[3]{D}x^{\frac{2}{3}}}{2} - \frac{27x^2}{700D} - \frac{3\cdot 2^{\frac{2}{3}}\sqrt[3]{3}x^{\frac{4}{3}}}{20\sqrt[3]{D}}$$

2.5 Numerical Use of the Puiseux Series Solution

Let us now use the truncated Puiseux solution to integrate numerically our ODE. First, fix $D = 2$:

[17]:
```
values[D] = 2
soln = R.evaluate (soln, values)
```

and evaluate the true solution (8) at $x_0 = .01$ and (say) $x_{\text{end}} = 1$:

[18]:
```
theta = var('theta')
def true_sol (x) :
    D_val = D.subs(values)
    theta_val = nsolve ((D_val/2) * (theta-sin(theta)) - x, theta, .1)
    return (D_val/2) * (1 - cos(theta_val))
```

[19]:
```
x0 = .01
xend = 1.
true_sol (x0), true_sol (xend)
```

[19]:
$$(0.0760417718282471,\ 1.35579714038883)$$

Let us now evaluate the Puiseux series at these points. As one might expect, the value obtained at x_0 is pretty accurate but becomes less precise at x_{end}:

[20]:
```
soln.subs({x:x0}).evalf(), soln.subs({x:xend}).evalf ()
```

[20]:
$$(0.0760417845217514,\ 1.35910982123678)$$

Now, the value y_0 provided by the Puiseux series at $x = x_0$ can be used as an initial value for any numerical ODE solver. For simplicity, let us use the one provided by the `scipy.integrate` package. It is an explicit Runge-Kutta method of order 4 with an embedded formula of order 5 used to control the error. The computed value y_{end} at $x = x_{\text{end}}$ is now much more precise:

[21]:
```
import math
def f (x,y) :
    D_val = D.subs(values)
    return math.sqrt (D_val/y - 1)
```

[22]:
```
from scipy.integrate import solve_ivp
y0 = soln.subs({x:x0}).evalf()
ode_sol = solve_ivp (f, [x0,xend], [y0], rtol=1e-8)
yend = ode_sol.y[0,-1]
abs (true_sol (xend) - yend)
```

[22]:
$$5.95431726146955 \cdot 10^{-7}$$

2.6 A Further Numerical Integration Problem

The initial point $(x_0, y_0) = (0, 0)$ is not the only problematic one. Indeed, at $(x_1, y_1) = (\pi\, D/2, D) = (\pi, 2)$, the partial derivative (7) becomes infinite also. At this point, solutions exist but are not unique.

Indeed, we have $\dot{y}(x_1) = f(x_1, y_1) = 0$ hence a horizontal tangent. One solution is the singular (constant) solution $y(x) = D$. The cycloid (which is an element of the general solution) is another solution passing through the same point.

From a numerical integration point of view, the problem shows up as follows. Consider the Runge-Kutta method (6). If (x, y) is close to (x_1, y_1) then, while computing k_2, the method attempts to evaluate $f(x + \frac{1}{2}h, y + \frac{1}{2}h\,k_1)$ with $y + \frac{1}{2}h\,k_1 > D$ which leads to the square root computation of a negative number.

Over this example, the numerical problem is easily overcome by integrating the following order 2 initial value problem. The ODE is the differential polynomial p_1 of section 1.5. The starting point is $(x_0, y_0, \dot{y}_0)$ where $x_0 = 0.01$, and y_0 is provided by the truncated Puiseux series, and $\dot{y}_0 = f(x_0, y_0)$ is provided by the brachistochrone equation.

$$2\,y\,\ddot{y} + \dot{y}^2 + 1 \;=\; 0\,. \tag{9}$$

In order to perform the numerical integration, the order two ODE is transformed into a system of two order one ODE. The unknown function y becomes a vector whose first coordinate is y and its second coordinate is $\dot{y}$. The result can be seen on Figure 2.

[23]:
```
import numpy as np
def g(x,y) :
    return np.array ([y[1], - (y[1]**2 + 1)/(2*y[0])], dtype=np.float64)

xend = 2*math.pi
yp0 = math.sqrt(D.subs(values)/y0 - 1)
ode_sol = solve_ivp (g, [x0,xend], np.array([y0,yp0],dtype=np.float64),␣
 ↪rtol=1e-8)
```

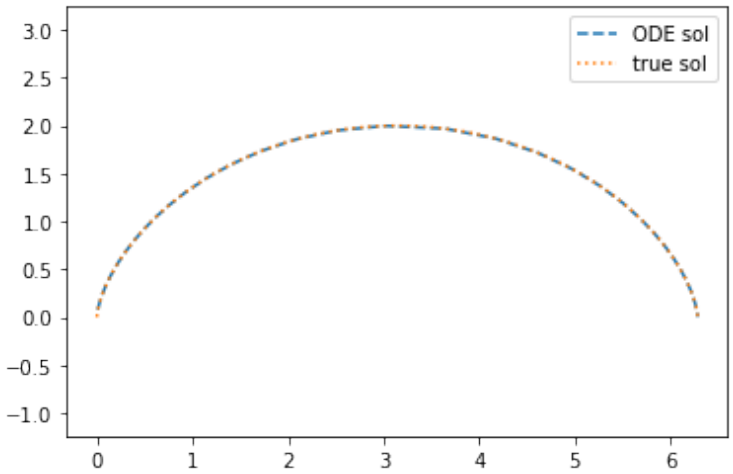

Figure 2: The curve obtained by numerical integration of ODE (9) and the true cycloid fit.

Here are the commands which produce Figure 2:

[24]:
```
import matplotlib.pyplot as plt
plt.axis('equal')
plt.plot (ode_sol.t, ode_sol.y[0], linestyle='dashed', label='ODE sol')
tplot = np.linspace (0, 2*math.pi, 100)
xplot = [ D.subs(values)/2*(theta - math.sin(theta)) for theta in tplot ]
yplot = [ D.subs(values)/2*(1 - math.cos(theta)) for theta in tplot ]
plt.plot (xplot, yplot, linestyle='dotted', label='true sol')
plt.legend ()
plt.show ()
```

References

[1] François Boulier and al. DifferentialAlgebra. https://codeberg.org/francois.boulier/DifferentialAlgebra.

[2] Ernst Hairer, Syvert Paul Norsett, and Gerhard Wanner. *Solving ordinary differential equations I. Nonstiff problems*, volume 8 of *Springer Series in Computational Mathematics*. Springer–Verlag, New York, 2nd edition, 1993.

[3] Elizabeth L. Mansfield. *Differential Gröbner Bases*. PhD thesis, University of Sydney, Australia, 1991.