Ordinary Differential Equations*

3D-Filmstrip knows how to calculate and display solutions of the initial value problem for first and second order systems of ordinary differential equations (ODE) in one, two, or three dependent variables.

Let us recall briefly what this means. We will be dealing with vector-valued functions x of a single real variable t called the “time”. Here x can take values in $\mathbb{R}$, $\mathbb{R}^2$, or $\mathbb{R}^3$. The problem is to find x from a knowledge of how x' depends on x and t (in the first order case) or a knowledge of how x'' depends on x, x', and t (in the second order case). Thus in the first order case the ODE we are trying to solve has the form $x' = f(x, t)$ and in the second order case it is $x'' = f(x, x', t)$.

In the first order case, the so-called Local Existence Theorem for First Order ODEs tells us that, provided the function f is continuously differentiable, given an “initial time” t_0, and an “initial position” x_0, then in some sufficiently small interval around t_0, there will be a unique solution $x(t)$ to the ODE with $x(t_0) = x_0$.

There is a similar local existence theorem for second order ODEs (which in fact is an easy consequence of the first order theorem). It says that given an initial time t_0, an initial position x_0, **and** an initial velocity v_0 then, in some

* This file is from the 3D-XplorMath project. Please see:

http://3D-XplorMath.org/

sufficiently small interval around t_0, there will be a unique solution $x(t)$ of the ODE with $x(t_0) = x_0$ and $x'(t_0) = v_0$.

Theory provides not only an abstract existence theorem, but also many explicit numerical algorithms for finding approximating solutions, in terms of the function f and the initial data. One of the all-time favorites for general purposes is the so-called Fourth Order Runge-Kutta Method, and this is the one that 3D-Filmstrip uses.

Although the overall approach to solving such ODEs is quite similar for first and second order ODEs and for the various dimensions, the details for giving initial conditions and for displaying solutions are different for each case, and for that reason instead of having a single ODE category, it turns out to be convenient to have six. The naming of these categories is fairly self-evident. For example, the ODE(1D) 1stOrder Category deals with the case that x take values in $\mathbb{R}$, and the equation is first order, while the ODE(3D) 2ndOrder Category deals with the case that x take values in $\mathbb{R}^3$, and the equation is second order.

For the ODE(1D) 2ndOrder category, the usual reduction to a first order system in two variables (x, u), where u represents x', is made via $(x, u)' = (u, f(x, u, t))$.

In the case of the ODE(2D) 1stOrder category, the orbit is drawn dotted, with a constant time interval between dots. This gives a valuable visual clue concerning the velocity at which the orbit is traced out, but if you wish to turn this feature off, just set Dot Spacing to zero using the ODE

Settings... dialog (see below).

In all the ODE categories, when you have chosen either a particular pre-programmed example (or set up your own, using the User Defined… feature) then as with the other categories you will first see a visualization of a default solution. This display will usually stop quickly on its own, but you can also click the mouse button (or type Command period) to stop it. You may then choose the item "ODE Settings…" in the Settings menu and this will allow you to set the various data the program needs to compute and display an orbit, namely:

a) the initial time,

b) the time-span,

c) the step-size (used in the Runge-Kutta method),

d) the initial value of x, and (in a second order case)

e) the initial value of x'.

Choosing Create from the Action Menu will then display the solution for these newly selected settings.

There is an ODE control panel that opens by default just below the main display window. This has buttons to do more easily things you can also do with the menus (Create, Erase, Continue, double or half the scale, and bring up the dialog to set initial conditions, step-size, time-span, and dottedness). In addition there are buttons for single-stepping the ODE forward or backward, and there is a read-out of the current time, position, and velocity. This control panel can be hidden using the Hide ODE Controls command of the Action menu (and then may be re-opened with the Show ODE Controls command).

The main display shows the evolution of an orbit in the phase space. By default, the program also shows projections of the orbit on the coordinate axes (using different colors to distinguish the projections). This display occurs in a second pane of the graphics window that opens automatically below the main pane. This pane can be hidden by choosing ”Hide Direction Fields” from the Action menu. There is a rectangular button at the right edge of the screen where the two panes meet. If you press on this button, the button itself will disappear and be replaced by a horizontal line. Drag the horizontal line to where you would like the new boundary between panes and release the mouse. (At least twenty percent of the total screen height must be devoted to each pane.)

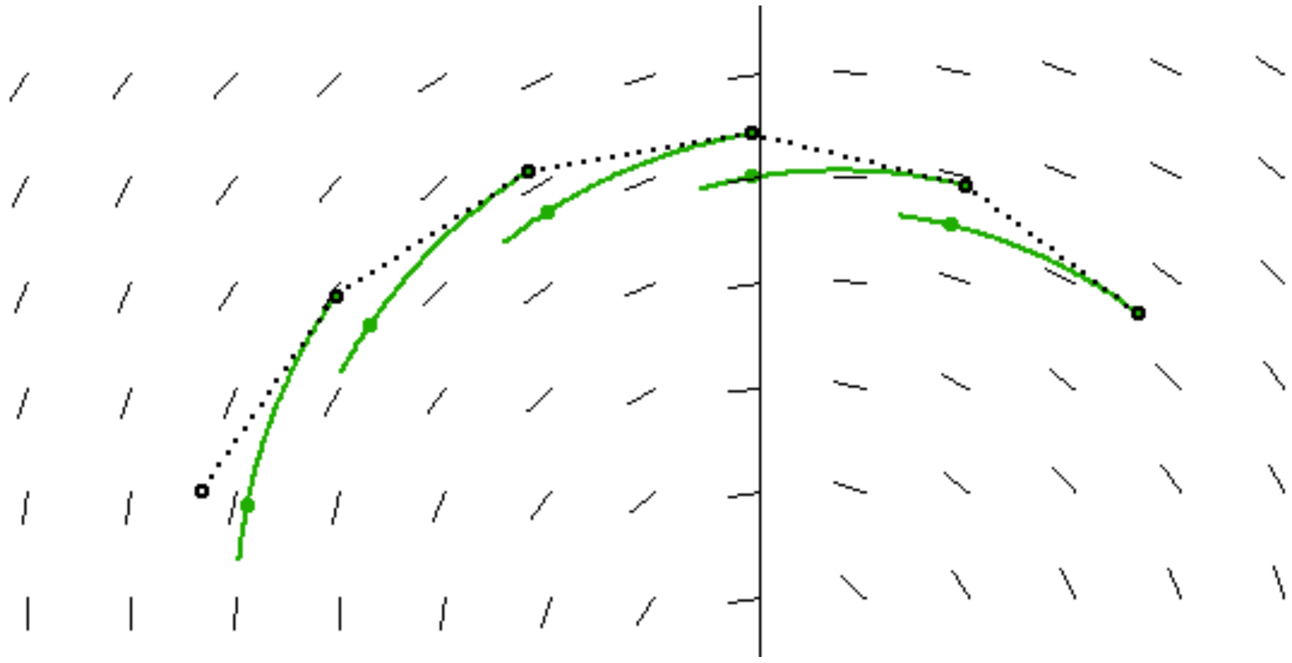
For first order ODE in one and two dimensions, the program by default displays the direction field defined by the current ODE. (Since second order ODEs in one variable are reduced to first order ODEs in two variables, the direction field is also shown in this case.) The direction field of a time dependent ODE is updated every few integration steps. (As far as I know, 3D-Filmstrip is the only publicly available program that shows direction fields for time-dependent ODE.) For the ODE(3D) 2ndOrder category there is also a direction field shown (when the display is in stereo) for the special case of a charged particle in a magnetic field—but be careful, the field shown is the magnetic field, **not** the direction of the Lorentz force acting on the particle.

It is fairly easy to do a rough "phase space analysis" by keeping the other data fixed and varying the initial values. To make this easier, in certain categories it is possible to choose the initial conditions using the mouse. For example, in the ODE(1D) 1stOrder category, each time you click the mouse in the window, the forward and backward orbits are drawn through that initial value. Similarly, clicking the mouse in the ODE(1D) 2ndOrder category and ODE(2D) 1st Order category also creates an orbit with the mouse point as initial condition. Surprisingly, something similar even works for the ODE(2D) 2ndOrder category. Here to choose an initial condition AND velocity, either select IC By Mouse [Drag] from the Action menu or type Control I. You may then click and drag the mouse to choose the initial position (click) and velocity (drag).

Numerical Methods for 1st Order ODEs

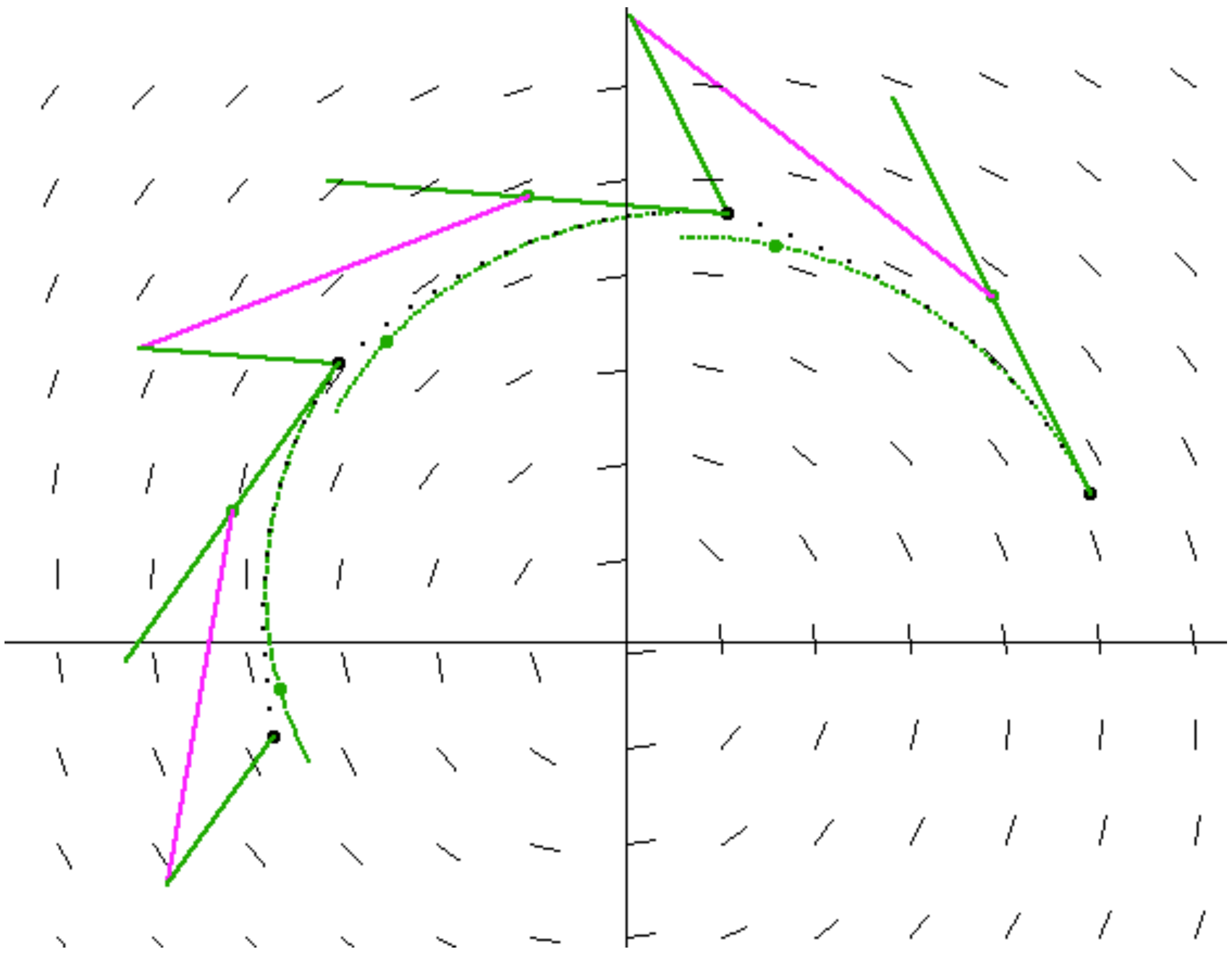
Numerical solutions are computed at a discrete set of time points $t_0, t_1, \ldots, t_n$ and usually one such time step of a method is described, i.e. how to get from $x(t_0)$ to $x(t_1)$. One then has to repeat this step as long as one wants to. The simplest of all methods is the EULER METHOD: From the initial data $t_0, x_0 = x(t_0)$ one computes first the
INITIAL DERIVATIVE: $x'(t_0) := f(x_0, t_0)$, then the
EULER STEP: $x(t_1) := x_0 + x'(t_0) \cdot (t_1 - t_0)$.

The following picture shows five such (large) Euler steps and the corresponding exact solutions for the same time interval. Clearly, if the exact solution curves happen to be convex, the Euler Step solutions stay outside and move farther away from the exact solution with each step.


Five Euler Steps

Recall that the tangent of a parabola wich is parallel to some secant touches the parabola in the middle of the secant interval. This suggests a substantial improvement over the Euler method, it is called the HALFSTEP METHOD:
STEPSIZE: $\Delta t := t_1 - t_0$
INITIAL DERIVATIVE: $x'_0 := f(x_0, t_0)$
HALF STEP TOWARDS MIDDLE: $x_m := x_0 + x'_0 \cdot \Delta t/2$
MIDDLE DERIVATIVE: $x'_m := f(x_m, t_0 + \Delta t/2)$
FINAL STEP: $x(t_1) := x_0 + x'_m \cdot \Delta t.$
Again we illustrate this method with a picture, but using a larger stepsize than in the Euler case. The magenta

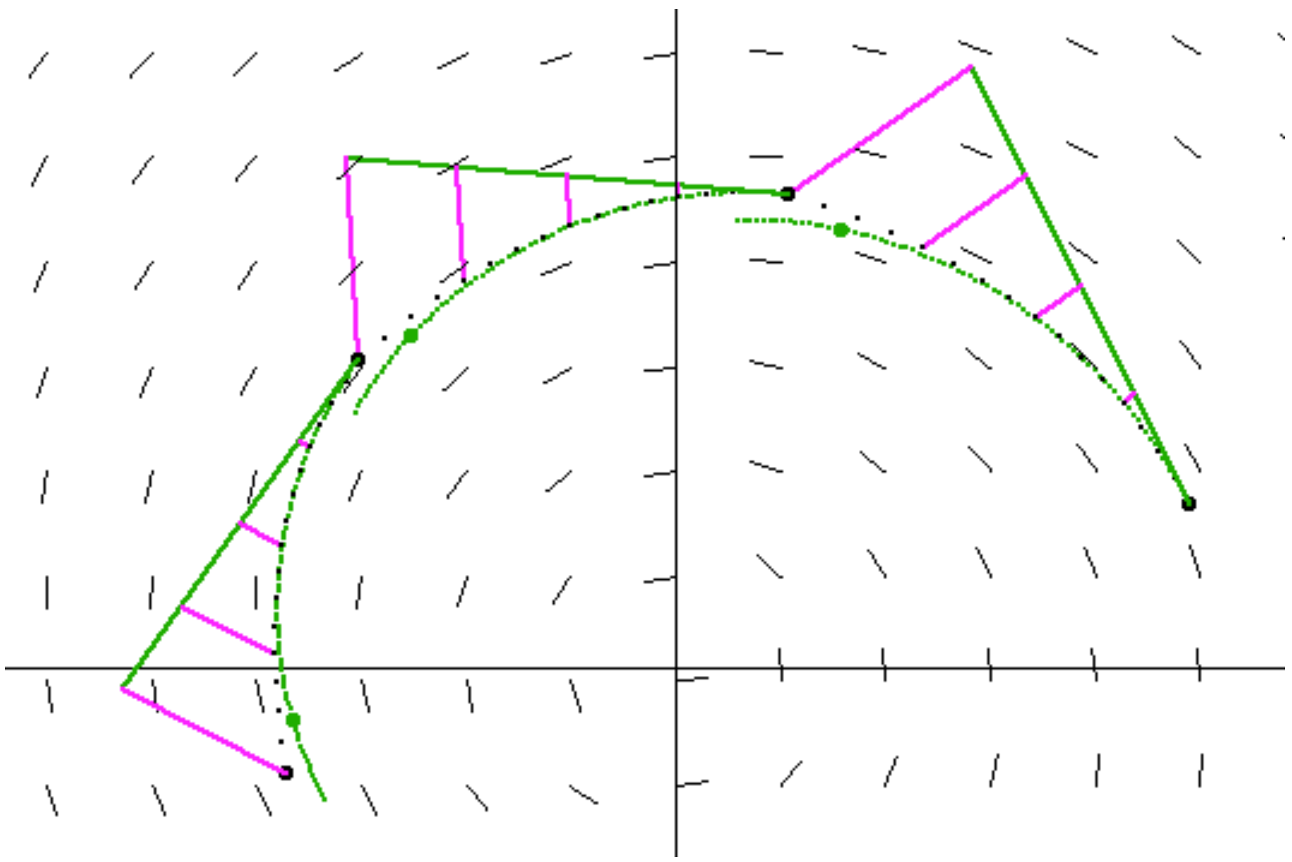
vector is the derivative at the approximate midpoint x_m multiplied by the stepsize, i.e. $x'_m \cdot \Delta t$. This vector is the difference between $x(t_1)$ and $x(t_0)$. One can see that this method follows the solution curve rather well, but traverses it too fast.



Three Halfstep Steps

Some first order ODEs have the property that the second initial derivative can be computed rather easily, more easily than differentiation of the ODE, $x'' = \frac{\partial}{\partial x} f \cdot x' + \frac{\partial}{\partial t} f$, suggests. In those cases one can use a second order Taylor step instead of the first order step used in Euler's method.

We call this the TAYLOR-2 METHOD:
INITIAL 1ST AND 2ND DERIVATIVE: x_0', x_0''
TAYLOR STEP: $x(t_1) := x_0 + x_0' \cdot \Delta t + x_0'' \cdot (\Delta t)^2/2$.
The endpoints of the magenta segments lie on the quadratic parabola $p(s) := x_0 + x_0' \cdot s + x_0'' \cdot s^2/2,\ s \in [0, \Delta t]$ (with the same initial tangent (green) as the exact solution). The method has the same order of precision as the previous one, so that one will often prefer the Halfstep method. But in the vicinity of a point where $f(x,t) = 0$, the Taylor-2 method is usually better.


Three Taylor-2 Steps

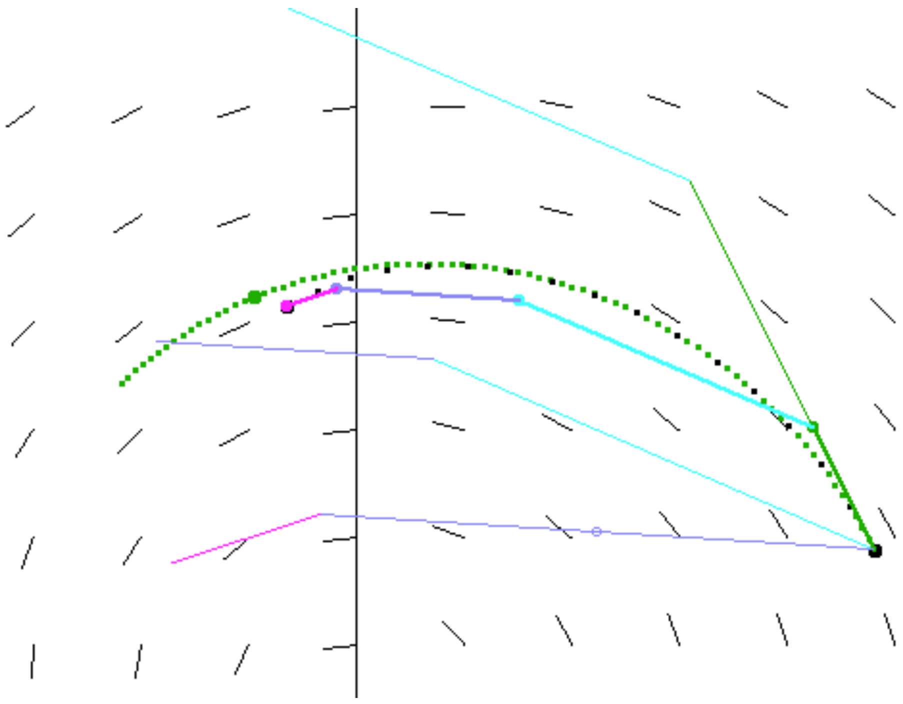
Next we try to explain the famous RUNGE-KUTTA METHOD. First we need to understand what numerical analysts call

an order k method. If we compute one Euler step from x_0, but with various stepsizes s, then all the computed points lie on the tangent $x_0 + x_0' \cdot s$. The difference to the exact solution is controlled by a bound B_2 on the second derivative of solutions near x_0 as: $\mathrm{error}(s) \leq B_2 \cdot s^2$. This is called a first order method.
For the Taylor-2 method these stepsize dependent numerical values lie on a parabola which was already mentioned: $p(s) := x_0 + x_0' \cdot s + x_0'' \cdot s^2/2$. The difference to the exact solution $x(s)$ is controlled in terms of a bound B_3 for third derivatives as: $\mathrm{error}(s) \leq B_3 \cdot s^3$. This is called a second order method.
How can one make such considerations work for the Half-step method? Clearly the approximate midpoint is a function of the stepsize: $x_m(s) = x_0 + x_0' \cdot s/2$ and therefore the final point also: $x_f(s) := x_0 + f\big(x_m(s), t_0 + s/2\big) \cdot s$. We may call this curve the method-curve. Computation shows that the exact solution at t_0 and the method-curve at $s = 0$ have the same first and second derivative: $x_0'(t_0) = x_f'(0)$, $x_0''(t_0) = x_f''(0)$. The error, therefore, is again $\leq B_3 \cdot s^3$, and the method is also called a second order method.
What we saw in these examples is true in general: Consider the initial data x_0, t_0 as fixed. The numerical value of the computation of one step will then only depend on the stepsize s, giving us the method-curve $x_f(s)$ for the considered numerical solution. If the first k initial derivatives of the exact solution agree with the first k initial derivatives of the method-curve, then we speak of an order k method.

The Runge-Kutta Method can be seen as a generalization of the halfstep method: Instead of computing two derivatives x_0', x_m', Runge-Kutta computes four and averages them for the final step.

initial derivative: $x_0' := f(x_0, t_0)$
1st intermediate point: $x_a(s) := x_0 + x_0' \cdot s/2$
1st intermediate derivative: $x_a'(s) := f(x_a(s), t_0 + s/2)$
2nd intermediate point: $x_b(s) := x_0 + x_a' \cdot s/2$
2nd intermediate derivative: $x_b'(s) := f(x_b(s), t_0 + s/2)$
3rd intermediate point: $x_c(s) := x_0 + x_b' \cdot s$
3rd intermediate derivative: $x_c'(s) := f(x_c(s), t_0 + s)$
derivative average: $x_{RK}'(s) := (x_0' + 2x_a' + 2x_b' + x_c')/6$
final step: $x_f(s) := x_0 + x_{RK}'(s) \cdot s$.

We can compute the first four derivatives of this method-curve and check that the above defines a 4th order method. It is one of the most celebrated ODE solving numerical methods. A visualisation of one Runge-Kutta step is:



The three straight segments from the initial point end at the three intermediate points x_a, x_b, x_c. Note how far they are apart from each other compared to the small error of the composite step. (The dotted green curve is the exact solution, the black dots lie on the method-curve.)

In 3D-XplorMath ODEs are not only objects of visualisations, they are an essential tool for many computations in the program. The ODE for elliptic functions f, namely:

$$f'(z)^2 = P(f(z)),\ \ P \text{ a polynomial of degree 3 or 4}$$

is an example where the second derivative can be more easily computed than the first: $f''(z) = P'(f(z))/2$. We therefore use a different 4th order method:
Initial Derivatives: x'_0, x''_0
Intermediate Point: $x_m(s) := x_0 + x'_0 \cdot s/2 + x''_0 \cdot s^2/8$
Intermediate 2nd Derivative: $x''_m(s) := P'(x_m(s))/2$
Final Step: $x_f(s) := x_0 + x'_0 \cdot s + (x''_0 + 2x''_m(s)) \cdot s^2/6$
Although zeros of the polynomial P are constant solutions of the first order ODE, the just described method can start to compute the non-constant function f at points z where $f'(z) = 0$ as initial value. A Runge-Kutta solution would stay constant.

R.S.P., H.K.