Application of Relational Interval Arithmetic to Ordinary Differential Equations

William J. Older

Computing Research Laboratory
Bell Northern Research

wolder@bnr.ca

August 1994

Abstract

Relational interval arithmetic can be useful in the numerical solution of ordinary differential equations and can offer several advantages which are difficult to achieve using traditional approaches. The first and most fundamental of these is correctness in the sense that interval techniques, by *retaining* the remainder term in Taylor expansions, can compute an inclusion of the true trajectory; both conventional truncation error and roundoff error can thus be replaced by an interval of uncertainty in the solution. The second advantage is symmetry: the same formulation solves not only initial value problems, but also final value problems, as well as problems in which there is partial information at both ends or along the trajectory. Third, it can aid in determining the stability and structural stability of solutions by permitting the initial conditions and any parameters to be imprecise, in which case it approximates the bundle of trajectories consistent with the boundary data..

1. Introduction

The conventional approaches to the numerical solution of ordinary differential equations have several well-known problems. The most fundamental of these is that these techniques introduce errors at every step of the procedure: truncation error which results from using a finite order approximation to a Taylor series, and roundoff error due to floating point arithmetic. Even though these errors may

be locally small, their total effect is cumulative, and the numerical solutions may therefore be *qualitatively* different from the true solutions. Another quite different set of problems is due the limitations of functional language: one can directly solve the initial value problem (or the final value problem) but other more complex boundary conditions require iteration of the integration procedures, with a great increase in the complexity of the code and the cost of solution. Iteration must also used to investigate structural stability issues when the equations contain parameters whose value is not known precisely, as is typical of real engineering and scientific problems.

This paper describes how these problems may in principle be overcome using the techniques of relational interval arithmetic, introduced in [1] and popularized by its implementation in BNR Prolog in 1988 [2,3,6] and later extended to the constraint language CLP(BNR) [4,5]. The method is a sound inferencing procedure over the reals, based on local propagation techniques and with careful control of rounding direction, for narrowing the initial interval values of a set of variables connected by general arithmetic relations. For concreteness, this paper uses the notation of CLP(BNR) Version 4.3 in which real variables are created using the syntax "`X:real(LB,UB)`" and arithmetic constraint relations are enclosed by braces {}, similar to Prolog III notation.

2. Differential Equations - Simple Prolog Framework

First we establish some notation. The differential equation will be written generically as:

$$dy/dx = f(x,y)$$

where y may be a vector, but for notational simplicity will be treated as a single value here. We are interested in this equation within a box (an approximate "flow box") defined by closed intervals for x and y and such that f is differentiable within this box. As usual, if f does not depend on y, the algorithms will reduce to quadratures.

As a Prolog predicate, we want something of the conceptual form:

```
integrate(Initial,Final,Flowbox,Control,Output).
```

For simplicity, we will express the differential equation as a "constraint generating" clause of form:

```
rhs(X, Y, F):- {F is Exp }.
```

where X is the independent and Y the dependent variable, `Exp` is an arithmetic expression in X and Y representing the right-hand-side of the d.e.. For simplicity we will limit ourselves to control strategies based on binary subdivision of the range of X, so `Control` is an integer indicating the number of doublings of the interpolation points; in particular, `Control=0` indicates no subdivision. The initial and final values of x are known precisely in advance and are represented as ordinary floats rather than intervals. The trajectory `Output` takes the form of a list of points along the curve along with their slopes, which we will treat internally as a difference list of form `Head/Tail`.

With these assumptions, the code takes on the following form:

```
integrate([X0,Y0],[X1,Y1],Flowbox, Control, Out):- integer(Control),
      rhs( X0,Y0,F0),
      rhs( X1,Y1,F1),
      EndOut=[[X1,Y1,F1]],
      $integrate(Control,[X0,Y0,F0],[X1,Y1,F1], Flowbox, Out/EndOut).

$integrate(0,Initial, Final, Flowbox, [Initial|Ps]/Ps):- !,
      step(  Initial, Final, Flowbox).

$integrate(Control,Initial, Final, Flowbox, L/E):- Control>0,
      Cn is Control - 1,
      $interpolate( Initial, Final, Flowbox, Middle),
      $integrate(Cn,Initial,Middle, Flowbox, L/M),
      $integrate(Cn, Middle, Final, Flowbox, M/E).
```

The function of the second clause of `$integrate` is to recursively insert a new point midway between the initial value of X and the final value of X, and to construct a new Y value at this point initialized from the flow box (which is assumed to be of the form of a standard "declaration" `real(L,H) )`, and to define the slope at that point.

```
$interpolate( [X0,Y0,F0],[X1,Y1,F1], Flowbox, [XM,YM,FM]):-
      XM is 0.5*(X0 + X1),
      YMs : Flowbox,
      rhs( XM,YM,FM).
```

To complete the implementation we then need only provide the predicate `step` relating the values and slopes at one point to those at an adjacent point.

3. Interval Eulerian Integration

In order to keep things as simple as possible to begin with, we start with Euler's method, which replaces the differential equation by the difference equation which has the initial value form

$$y_1 - y_0 = f(x_0,y_0)\ (x_1 - x_0).$$

This is, of course, fundamentally incorrect and only provides a local approximation to the solution: it can thought of as arising from approximating the correct integral form:

$$y_1 - y_0 = \int_X f(x,y)\ dx,$$

where the X denotes the interval $[x_0,x_1]$. Since

$$m\ (x_1 - x_0) \leq \int_X f(x,y)\ dx \leq M\ (x_1 - x_0)$$

whenever $m \leq f(x,y) \leq M$ over interval $X_{01}=[x_0,x_1]$ and since the machinery of interval arithmetic can compute the natural inclusion $f(x,y)\ \varepsilon\ f(X_{01},Y)$ we get the inclusion

$$\int_{X01} f(x,y)\ dx\ \varepsilon\ f(X_{01},Y)(x_1 - x_0)$$

for Y *any* valid range estimate for y over interval X_{01}. In effect, we are using the natural interval extension of f to compute bounds from which we can generate an inclusion for Y1. This leads to the following set of interval relations:

$$F\ \varepsilon\ f(X_{01},Y_{01})$$
$$Y_{01} - Y_0 == F * DX$$
$$Y_1 - Y_0 == F * dx.$$

where dx is the precise value (x1-x0) and DX is the interval [0,dx]. The inclusion relation defines the range F of possible slopes between x0 and x1, and interacts with the first equation to narrow Y_{01}, and hence the F, until a stable values are reached. (By using an inclusion, we block any narrowing propagating from $Y_1 - Y_0$ to F and hence possibly to X_{01} or Y_{01}.) Then the second equation extrapolates the resulting slope F to the end of the interval in order to bound Y_1.

This would lead to the following code for the integration step:

```
step( [X0,Y0,F0], [X1,Y1,_], Flowbox):-
            X01: real(X0,X1),
            Y01 : Flowbox,      % initial range of Y in interval X01
            Dx is X1 - X0,      % assumed dx>0
            DX: real(0,Dx),     % or: U:real(0,1), DX is U*Dx,
            { Y01 - Y0 == F*DX,  % "slope predictor"
              Y1  - Y0 == F*Dx }.% extrapolate
```

It is apparent that one is getting a correct trajectory inclusion and that the algorithm works symmetrically in both directions. However, with such a low-order method the interval of uncertainty is very large and only improves slowly with subdivision, so this particular algorithm is impractical

4. Trapezoidal integration- Geometric version.

To improve matters we can use a second order inclusion in the integration step. Thus we return to the correct equation

$$y_1 - y_0 = \int_X \mathbf{f}(x,y)\ dx,$$

and try to refine our estimate of the definite integral. In this section we will use elementary geometric argument, and in the next section we will develop an alternative approach. If we look at a graph of f (regarded as a function of x alone *along the true trajectory)* we see that the Eulerian interval approximation computes the area under it as the interval $[m\Delta x, M\Delta x]$, with an "uncertainty" $(M-m)\Delta x$ where m and M are bounds of f. Geometrically speaking, trapezoidal integration uses the average of the values of f at the beginning ($f_0=f(x_0,y_0)$) and end ($f_1=f(x_1,y_1)$) of the interval. If the total derivative of f with respect to x is constant in the interval, this result is obviously exact. But in general there is an error represented by the area between the graph of f and this straight line interpolation.

If we have bounds [m,M] for the total derivative of f, f is now constrained to lie within the parallelogram P formed by the linear equations:

$$m(x - x_0) \le y(x)-x_0 \le M(x-x_0)$$
$$m(x_1 - x) \le y(x)-x_1 \le M(x_1-x)$$

for $x_0 \le x \le x_1$. Since the average slope line segement from y_0 to y_1 line bisects the parallelogram, the true value of the integral is bounded by $(1/2)*(f_0+f_1)*\Delta x \pm 1/2$ area(P).

It remains to calculate the area of the paralleogram P as the product of its base and height. With $\mu = (y_1 - y_0)/\Delta x$ being the average slope, and assuming that μ is greater than or equal to the average of M and m, the height h can be computed as

$$h = M\,\Delta x - \mu\,\Delta x = (M- \mu)\,\Delta x.$$

The reverse case leads eventually to the same formula. Similarly the base b can be computed from

$$y_0 + Mb = y_p = y_1(\Delta x - b)$$

so $(M - m)b = (\mu - m)\,\Delta x$

and finally $b = \Delta x\,(\mu - m)/(M - m)$.

Thus the area of P is given in the symmetric form:

$$\text{area} = \Delta x^2\,(\mu - m)(M - \mu)/(M - m).$$

This can be expressed more conveniently by introducing ρ $(0 \le \rho \le 1)$ defined by $\mu = \rho m + (1-\rho)M$, as

$$\text{area} = \rho(1-\rho)\,(M - m)\,\Delta x^2 .$$

Note that the error vanishes whenever $\rho = 0$ or $\rho = 1$, since the parallelogram degenerates into a line segment, implying that the graph of f is a straight line. This formula gives a maximum error (half of the area), which occurs at $\rho = 0.5$, of

$$\text{error} = \pm(1/4)\,(1/2)(M - m)\,\Delta x^2 .$$

5. Trapezoidal Integration by Taylor Series

In this section we will develop the second order error estimate by analytical means. We assume that f has a continuous total derivative with respect to x. in the interval $[x_0, x_1]$. If we expand y in a Taylor series with remainder about $x = x_0$ we get exactly

$$y(x) = y_0 + f_0\,(x - x_0) + {}_0R_2$$

and about $x = x_1$ we get exactly

$$y(x) = y_1 + f_1\,(x - x_1) + {}_1R_2$$

where ${}_0R_2$ and ${}_1R_2$ are the respective remainder terms. Taking the difference of the first evaluated at $x = x_1$ and the second evaluated at $x = x_0$ gives the identity

$$y1 - y0 = (1/2)(f_0 + f_1)(x_0 - x_1) + (1/2)({}_0R_2 - {}_1R_2) .$$

We recognize the first term on the right as the truncated trapezoidal formula. So it remains to generate an interval estimate of the remainder "error" quantity $E = (1/2)({}_0R_2 - {}_1R_2)$. To do this, it is helpful to review the derivation of the remainder term. From

$$y(x) = y_0 + \int y'(t)\,dt,$$

where the integral is from x_0 to x, we can apply integration by parts to get

$$y(x) = y_0 + \left[\, y'(t)(t-x) - \int (t-x)\,y''(t)\,dt \,\right],$$

where the evaluation is from t=x0 to t=x, giving

$$y(x) = y_0 + y'(x_0)(x - x_0) - \int_{[x0,x]} (t-x)\,y''(t)\,dt .$$

Since $y'(x_0) = f(x_0, y_0) = f_0$, this implies that

$${}_0R_2 = \int_{[x0,x1]} (x_1 - t)\,f'(t)\,dt$$

and similarly $_1R_2 = \int_{[x1,x0]} (x_0\text{-}t)\, f'(t)\, dt$.

Then

$$E=(1/2)(\ _0R_2 - {}_1R_2) = (1/2)\int_{[x0,x1]} (x_0 + x_1 - 2t)\, f'(t)\, dt$$

$$= \int_{[x0,x1]} ((x_0 + x_1)/2 - t)\, f'(t)\, dt.$$

We estimate this integral by breaking it into its two parts at the midpoint $xm=(x_0 + x_1)/2$ and estimating each part *using the same bounds:*

$$E = \int_{[x0,xm]} (x_m - t)\, f'(t)\, dt - \int_{[xm,x1]} (t\text{-}x_m)\, f'(t)\, dt$$

so $E \ \varepsilon\ (1/8)[m,M]\ (x_1 - x_0)^2 - (1/8)\ [m,M]\ (x_1 - x_0)^2$,

since $x_1 - x_m = x_m - x_0 = (x_1 - x_0)/2$, where [m,M] are bounds for f' over $[x_0,x_1]$. Thus the interval error term can be expressed as

$$E = (1/8)\ [m\text{-}M,M\text{-}m]\ (x_1 - x_0)^2$$

or as

$$E = (1/4)\ \delta(R)\ (x_1 - x_0)^2$$

where the symmetric interval $\delta(R)$ (:= (R - R)/2) represents the width of interval R=[m,M]. (Using separate bounds for each half is usually only slightly better, but would double the cost.) Note that this agrees with the result of the previous section.

This sort of analysis can easily be extended to higher order Taylor expansions. However, the usual rapid growth in complexity of formulae for higher-order convective derivative leads to increasingly poor intrinsic narrowing behaviour, and the increasingly pessimistic estimation of the remainder term at higher orders suggests that in most cases one should keep to the lower order methods and subdivide the x-range instead. With this trapezoidal method in initial value problems, each additonal subdivision typically halves (or better) the cumulative error/uncertainty in the final value, until one reaches a point where the increases in roundoff errors cancel reductions in truncation errors.

6. Implementation

Using the same ideas and notation as was used earlier we can easily implement the integration step of the previous two sections as:

```
step( [X0,Y0,F0], [X1,Y1,F1], Flowbox):-
            X01: real(X0,X1),
            Y01 : Flowbox,     % range of Y in interval DX
            Dx is X1 - X0,
            DX : real(0,Dx),
```

```
df( X01,Y01, R),     % f' along traj over interval X01
{ FM is (F0 + F1)/2, % average slope
  Y01 - Y0 == FM*DX + (1/4) d(R) DX**2,
  Y1 - Y0 ==  FM*Dx + (1/4) d(R) Dx**2 }.
```

Here we used the notation d(R) to represent the continuous symmetric width or "delta" of interval R. Note that if Df/dx is constant over the interval, d(R)=0. The only new feature that is required is a predicate d f to compute the derivative of f with respect to x along the trajectory, i.e. the convective derivative given by $D_x y' = \partial_x f + f\partial_y f$.

This formulation has several important properties that make it qualitatively different from conventional integration techniques. The first is that the solution is given by intervals of the dependent variables at each interpolated value of the independent variable, and these intervals are guaranteed to include the true trajectory of the original differential equation. That is, conceptually the notion of "truncation error" as well as the notion of "floating point error" for precisely posed problems have been replaced by the notion of an "interval of uncertainty." The fact that these intervals include the true trajectory means that if one generates several solutions for the same problem using different control strategies, the several solutions can be merged by intersection of these intervals.

Secondly, since the relations making up the constraint network are treated symmetrically by the CLP machinery, we are free to impose any boundary conditions we like. In particular, classical two-point boundary conditions in which some of the dependent variables have initial conditions specified while others have final conditions specified are handled by the same code. More exotic non-classical situations, in which some boundary conditions may be specified by e.g. inequalities at some point along the trajectory, can also be handled.

Finally, stability issues due to imprecise boundary conditions as well as structural stability issues due to imprecise parameters are also handled by the same formulation. These also are reflected as increased uncertainty in the solution intervals. Although this effect is almost always overly pessimistic, it can be useful to suggest when stability is likely to be a serious issue and when it is not. In the cases where stability analysis is needed, investigating extreme trajectories can be done directly by narrowing the boundary intervals without recomputing from scratch.

Practically speaking, the drawbacks of this approach seem to be the large amount of storage required to hold the constraint network, and the need to specify an initial "flowbox" which is tight enough to narrow properly.

Acknowledgement

I wish to express my thanks to Dr. Reinhard Skuppin of FAW, whose investigations motivated this article, for his comments on an early draft and for several contributory discussions.

References

[1] J.G. Cleary, Logical Arithmetic. *Future Computing Systems*, 2(2):125-149, 1987.

[2] *BNR Prolog User's Guide*, 1988.

[3] *BNR Prolog Reference Manual*, 1988.

[4] F. Benhamou and W. Older, Applying Interval Arithmetic to Real, Integer and Boolean Constraints. *Journal of Logic Programming*, 1993. (Submitted)

[5] F. Benhamou, W. Older and A. Vellino, Constraint Logic Programming on Boolean,Integer and Real Intervals. *Journal of Symbolic Computing*, 1994. (Submitted)

[6] W. Older and A. Vellino, Extending Prolog with Constraint Arithmetic on Real Intervals. *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 1990.

[7] W. Older and A. Vellino, Constraint Arithmetic on Real Intervals. *Constraint Logic Programming:Selected Research*, F. Benhamou and A. Colmerauer eds., The MIT Press, Cambridge, MA,1993.

[8] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C*, Second Edition, Cambridge University Press, 1988.