The Predicted Sequential Regularization Method for Differential-Algebraic Equations

Colin Macdonald

(email: 028741m@acadiau.ca)

May 10, 2000

Abstract

Differential-algebraic equations (DAEs) can be used to describe the evolution of many interesting and important systems.

This report will involve the investigation of numerical methods for solving DAEs. In particular, a new numerical method called the Predicted Sequential Regularization Method (PSRM) will be examined.

It will be shown that the concept of the *index* of a DAE leads to a simple and intuitive method for the solution of DAEs that unfortunately does not work particularly well. Through a simple example, the benefits of the PSRM over this intuitive method will be demonstrated. The implementation of the PSRM will be discussed including the ideas of *embedded Runge-Kutta* and *continuous Runge-Kutta*.

Contents

[1] Consult [2] for more information regarding the method outlined here

1 Introduction

Differential-algebraic equations are systems of differential equations with constraints on their solutions. The generic DAE is

$$x' = f(x,t) - B(x,t)\lambda, \tag{1}$$

$$0 = g(x,t). \tag{2}$$

More specifically, a system can be identified as a DAE if:

1. The derivatives of at least one of the unknown functions of t appears in one or more of the equations (we will use x to collectively represent these variables).

2. At least one of the unknown functions appears only as algebraic quantities (we will use λ for these variables).

3. One or more *constraint equations* (2) describe a manifold on which solutions for the differential variables (i.e., x) must lie (we will use $0 = g(x,t)$ for the constraint equations).

For example, consider the system

$$x' = Ax + q + B\lambda,$$

$$0 = Cx + r,$$

where x and λ are unknown functions of some independent variable, say t; q and r are known functions of t; and A and B are constant matrices. By definition, this is a DAE because x' appears in the system, λ' *does not* appear in the system, and values of x are constrained by $0 = Cx + r$.

DAEs have many important applications such as describing the dynamics of multi-body systems. Multi-body systems are made up of components whose motions are interconnected such as the mechanical problems in the field of robotics. When such systems are modeled with DAEs, the constraint equations are used to describe these connections between the components. In some sense, the constraint equations, combined with the algebraic variables (λ), impose the forces which hold the system together.

2 Index of a DAE

We define the *index* of a DAE as the number of times one needs to differentiate the constraint equation to obtain a ODE for λ. In our example:

$$\begin{aligned}
0 &= Cx + r \\
0 &= Cx' + r' \\
0 &= C(Ax + q + B\lambda) + r' \\
0 &= C(Ax' + q' + B\lambda') + r'' \\
CB\lambda' &= -CAx' - Cq' - r'' \\
\lambda' &= (CB)^{-1}(-CAx' - Cq' - r'').
\end{aligned}$$

Hence, we have transformed the original DAE into the ODE

$$\begin{aligned}
x' &= Ax + q + B\lambda, \\
\lambda' &= (CB)^{-1}(-CAx' - Cq' - r'')
\end{aligned}$$

by differentiating twice, so our example is an *index-2* DAE.

3 Solving DAEs by Repeated Differentiation

The notion of index of a DAE suggests the following method of solution: through repeated differentiation, transform the problem into a system of ODEs and then use an established numeric method to solve the new system.

As the following example will illustrate, this simple method is far from ideal.

4 The Pendulum Problem

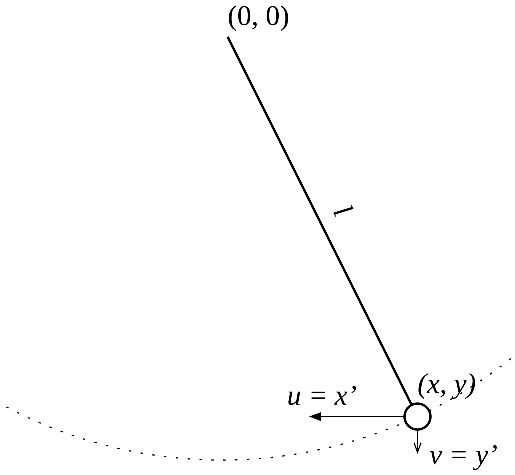

Figure 1: A simple pendulum

Consider a simple pendulum (Figure 1) of length l suspended from the origin such that the ball (assume unit mass) is at the point (x, y). Assuming the drag is negligible, then the motion of the ball, independent of the arm, would be described in Cartesian coordinates by $x'' = 0$ and $y'' = -g$. In terms of vectors, we would express this motion as the system

$$q' = p,$$

$$Mp' = f,$$

where $q = \binom{x}{y}$ is the position of the ball, $p = \binom{u}{v}$ is its velocity, M is its mass (the 2x2 identity in this case), and $f = \binom{0}{-g}$. However, when the motion of the ball is constrained by the arm (of negligible mass) of the pendulum (i.e., $l^2 = x^2 + y^2$), its motion can be described by the DAE

$$
\begin{aligned}
q' &= p, \\
Mp' &= f - G^T\lambda, \\
0 &= g(q),
\end{aligned}
$$

where $g(q) = 0$ is the constraint equation, $G = \frac{\partial g}{\partial p}$, and $G^T\lambda$ is the force that maintains the constraints. Furthermore $G^T\lambda = \binom{2x}{2y}\lambda = \binom{x}{y}2\lambda$ and, without loss of generality, we can replace 2λ with λ and thus, rewriting this system in a scalar sense yields the DAE

$$
\begin{aligned}
x' &= u, \\
y' &= v, \\
u' &= -\lambda x, \\
v' &= -\lambda y - g, \\
0 &= x^2 + y^2 - l^2.
\end{aligned}
$$

Using the transformation $x = l\cos\theta, y = l\sin\theta$, it can be shown that these equations are equivalent to the more familiar polar coordinates equation, $\theta'' + \frac{g}{l}\sin\theta = 0$.

The pendulum problem is an index-3 DAE, i.e., after the three differentiations, it can be written as a system involving only differential equations:

$$
\begin{aligned}
x' &= u, \\
y' &= v, \\
u' &= -\lambda x, \\
v' &= -\lambda y - g, \\
\lambda' &= \frac{-4\lambda x u - 4\lambda y v - 3 g v}{x^2 + y^2}.
\end{aligned}
$$

We can then solve this system using an ODE solver such as `matlab`'s `ode45`. Figure 2 shows a trace of the path taken by a pendulum released from rest at point $(1, 0)$ using this technique of repeated differentiation (RD). Clearly, this is an unreasonable

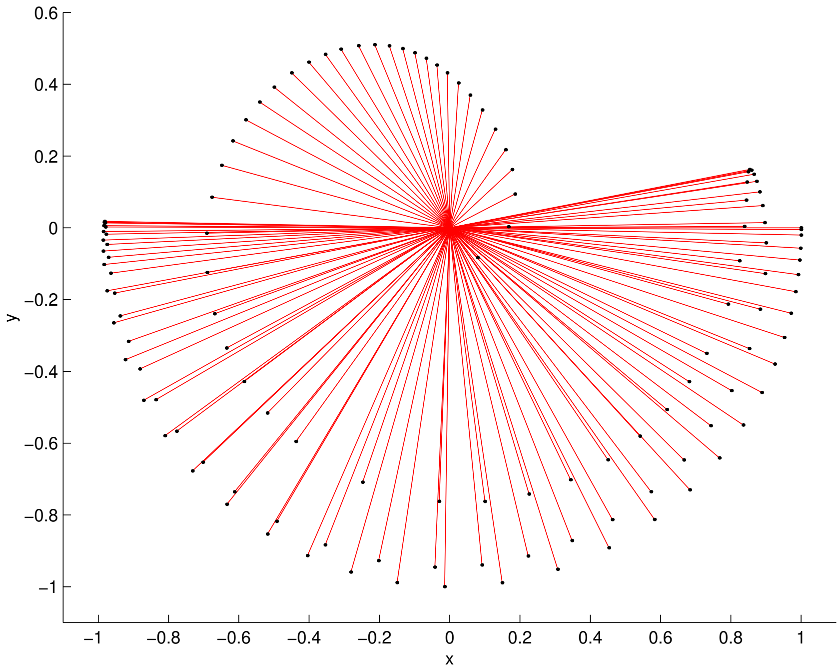

Figure 2: Path of a pendulum using RD, $t_0 = 0, t_f = 5, \mathrm{tol} = 10^{-3}$

simulation — the length of the arm of the pendulum has changed over time so this path is physically impossible!

It seems this method of solving DAEs by reducing them to ODEs causes *drift* in the values of the variables. We define the drift at some time t as the value of $g(q)$ (i.e., the RHS of the constraint equation). Or, in the context of our pendulum problem, how much $x^2 + y^2$ differs from l^2.

It is perhaps prudent to point out that there is nothing "wrong" with the model, either in its original DAE form or the transformed ODE form. If it were possible for `ode45` to solve the problem without making small roundoff errors at each time-step, then it would output the correct path. However, what we have done is converted a stable problem into an unstable one; that is, the aforementioned small errors tend to blow up over time.

5 The Sequential Regularization Method

5.1 Baumgarte Stabilization Technique

Recall the generic differential-algebraic equation

$$x' = f(x,t) - B(x,t)\lambda, \tag{3}$$

$$0 = g(x,t), \tag{4}$$

where the constraint equation defines a manifold on which solutions should lie. Unfortunately, written in this form ($0 = g(x,t)$), the constraint manifold is inherently unstable. In the intuitive repeated differentiation technique presented above, this instability manifests itself in the drift observed in the solution to the pendulum problem.

In the 1970's, Baumgarte [1] noted this and worked on the replacing the constraint equation with an exponential decay ODE,

$$\alpha_1 \frac{\mathrm{d}}{\mathrm{d}t} g(x,t) + \alpha_2 g(x,t) = 0 \qquad \alpha_1 > 0, \alpha_2 \geq 0,$$

with the intent that this would make the constraint attractive and thus improve it stability.

Unfortunately, in practice, he was not able to get this work because it was difficult to choose the parameters α_1 and α_2. It turns out they are dependent on one's choice of h.

5.2 The Sequential Regularization Method

From Baumgarte's work, the original Sequential Regularization Method (SRM) was derived:

$$x_s' = f(x_s,t) - B(x_s,t)\lambda_s \tag{5}$$

$$\text{where} \quad \lambda_s = \lambda_{s-1} + \frac{1}{\epsilon} E(x_s,t) \left[\alpha_1 \frac{\mathrm{d}}{\mathrm{d}t} g(x_s,t) + \alpha_2 g(x_s,t) \right], \tag{6}$$

where ϵ is the so-called *regularization parameter* and $E(x_s,t)$ is the *regularization matrix*. Common choices for $E = E(x_s,t)$ are $E = (GB)^{-1}$, $E = (GB)^T$, or in the special case of multi-body system, $E = I$ (because $G = B^T$).

The idea behind the SRM is that one solves two *sequences*, $\{x_s(t)\}$ and $\{\lambda_s(t)\}$, which ideally converge point-wise to $x(t)$ and $\lambda(t)$ respectively as s increases. Starting with the initial conditions of the problem as $x_0(0)$, and using some initial guess for $\lambda_0(t)$, one simultaneously solves (6) for $\lambda_1(t)$ and (5) for $x_1(t)$ using Forward Euler or, preferably, one of the many forms of Runge-Kutta. This process is repeated $m \in \mathbb{N}$ times to allow for convergence.

It turns out that $\alpha_1 = 0$ and $\alpha_2 = 1$ are good choices for the parameters. Thus, sequences (5) and (6) become

$$x_s' = f(x_s, t) - B(x_s, t)\lambda_s \tag{7}$$

$$\lambda_s = \lambda_{s-1} + \frac{1}{\epsilon} E(x_s, t) g(x_s, t). \tag{8}$$

Note that (8) rearranges to

$$\epsilon E^{-1}(x_s, t)\left(\lambda_s(t) - \lambda_{s-1}(t)\right) = g(x_s, t), \tag{9}$$

which appears similar to the constraint equation if the LHS of (9) is small enough. Thus the purpose of the regularization parameter ϵ and the regularization matrix $E(x_s, t)$ is to make the LHS of (9) smaller.[2] However the beauty of the SRM with $\alpha_1 = 0$ and $\alpha_2 = 1$ is that it can be shown that $x_s(t) - x(t) = \mathcal{O}(\epsilon^s)$ and $\lambda_s(t) - \lambda(t) = \mathcal{O}(\epsilon^s)$, and thus it is not necessary to start with an extremely small ϵ; that is, one can start with a not-so-small ϵ of say 10^{-2} and as the SRM is iterated, the $x_s(t)$ will converge to the true $x(t)$.

See [3, 4] for addition information on the SRM.

The larger the value we choose for m, the longer the SRM will require to run. Clearly, we wish to choose as small a value as possible; which leads to the concept of the Predicted SRM that we will examine next.

6 The Predicted SRM

With the Predicted Sequential Regularization Method (PSRM), we try to use $m = 1$ (that is, solve (7) and (8) only once) by making a better guess at $\lambda_0(t)$. In order to

[2] see [5, 6, 7, 8, 9] for more information on the subject of regularization.

do so, it turns out that we need high-order approximations of $y(t)$ for all values of t where $f(t, y)$ is evaluated by the Runge-Kutta method. In fact, in section 9, we see that so-called *dense output* or *continuous* Runge-Kutta methods can be obtained. These method provide a continuous approximation at an order equivalent to that of the Runge-Kutta method on which they are based. And they do so relatively cheaply (i.e. without many more function evaluations!)

As we will see in section 7, we can solve our pendulum problem with PSRM without worrying too much about section 9.

7 Reconsidering the Pendulum Problem

We can use the Predicted Sequential Regularization Method to re-solve the pendulum problem from section 4. The `fortran` code used to produce these results uses Heun's Method, a 2-stage explicit Runge-Kutta method which only samples the function at the end points of the step. Because of this, we can use PSRM without deriving a continuous extension of Heun's Method. Also, this implementation has a constant step-size (i.e. h is specified as a parameter) and thus does not use automatic step-size control (section 8).

Figure 3 corresponds to figure 2 and shows the path of the pendulum from $t_0 = 0$ to $t_f = 5$ as returned by the PSRM. Clearly this output is much more reasonable as the length of the pendulum has not visibly changed as it did with the repeated differentiation output in figure 2.

By decreasing the tolerance in `ode45`, we can obtain a much better estimate of the path of the pendulum (see figure 4) while still using repeated differentiation (RD). As this looks much better than figure 2, one might be tempted to assume that repeated differentiation will work provided the tolerance is sufficiently small.

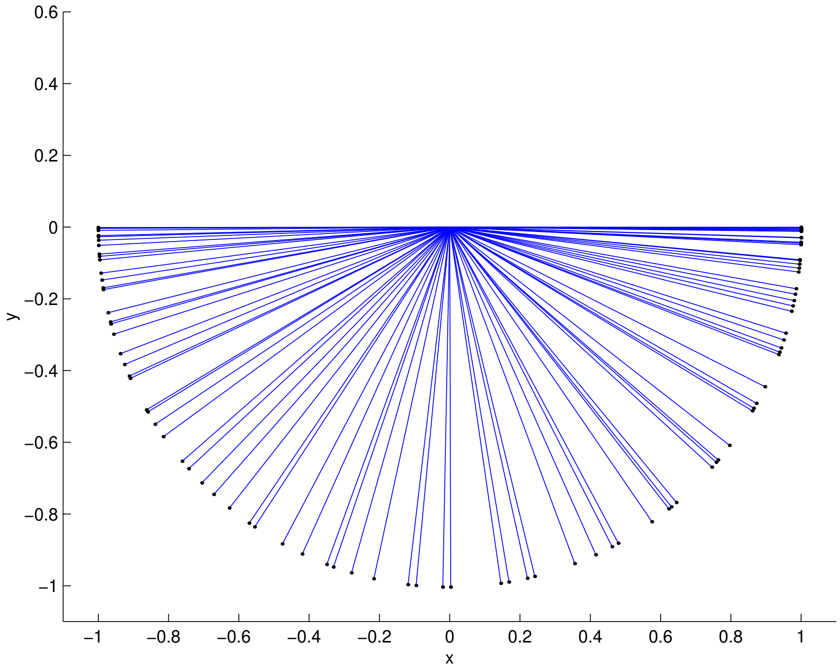

Figure 3: Path of the pendulum using PSRM, $t_0 = 0, t_f = 5, m = 1, h = .01$

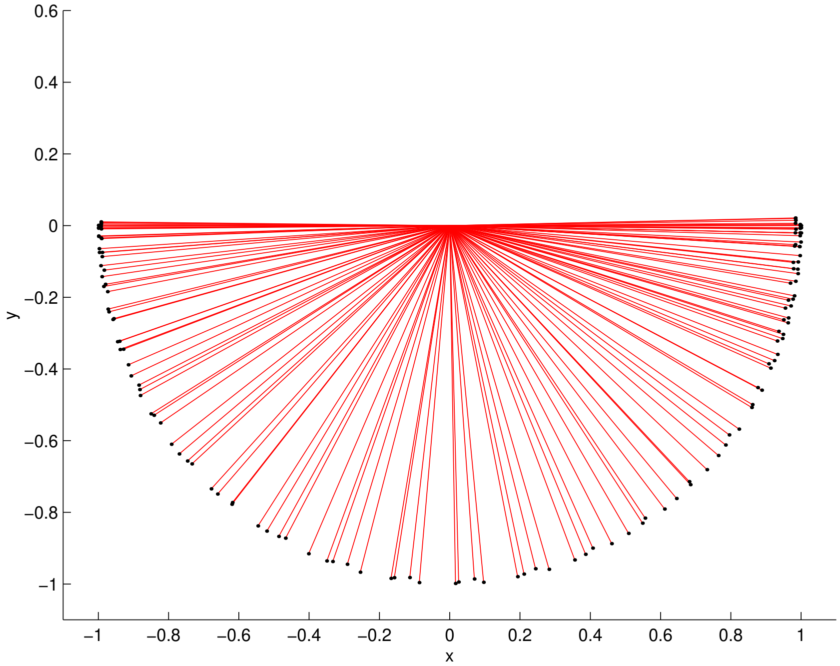

Figure 4: Path of the pendulum using RD, $t_0 = 0, t_f = 5, \mathrm{tol} = 10^{-4}$

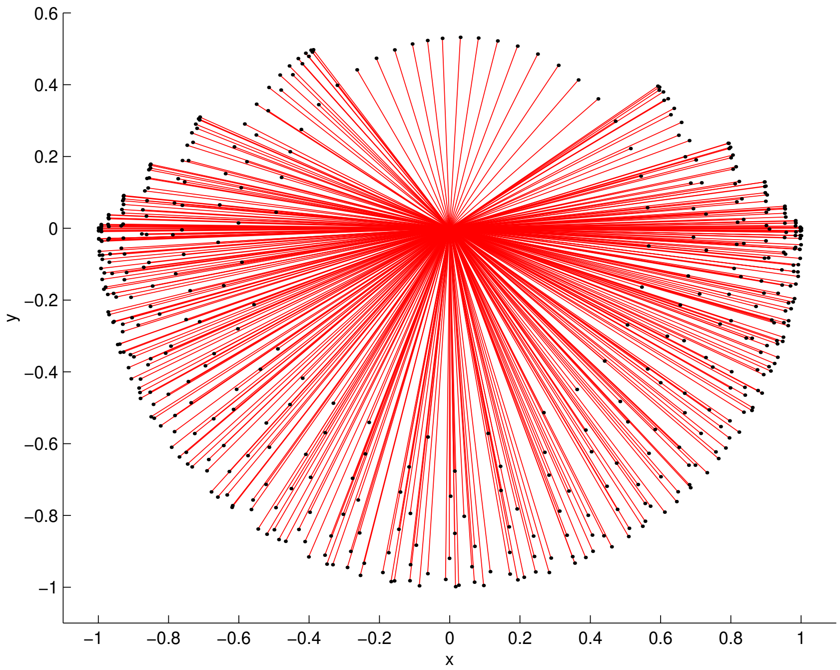

Figure 5: Path of the pendulum using RD, $t_0 = 0, t_f = 17, \mathrm{tol} = 10^{-4}$

However, as shown by figure 5, we can simply choose a higher value for t_f and drift will again become apparent. We can only play this game for so long; for some value for t_f, we will no longer be able to choose a tolerance small enough because of roundoff error. Thus, we can only model the path of the pendulum for (small) finite time and that time is dependent on ϵ_{MACH}. This is not ideal, for we would like to be able to model the path without such restrictions.

The tolerance required for the PSRM to correctly model the pendulum, on the other hand, is not a function of t_f — this is a sign that the small roundoff errors made at each time step are decaying away and thus not blowing up over time. Clearly, this is preferable to the repeated differentiation method as we can model the path of the pendulum for as long as is desirable without changing the values of parameters. Figure 6 shows the path of a pendulum from $t_0 = 0$ to $t_f = 20$ as modeled by PSRM using exactly the same parameters as in figure 3 (except that t_f is 20 instead of 5).

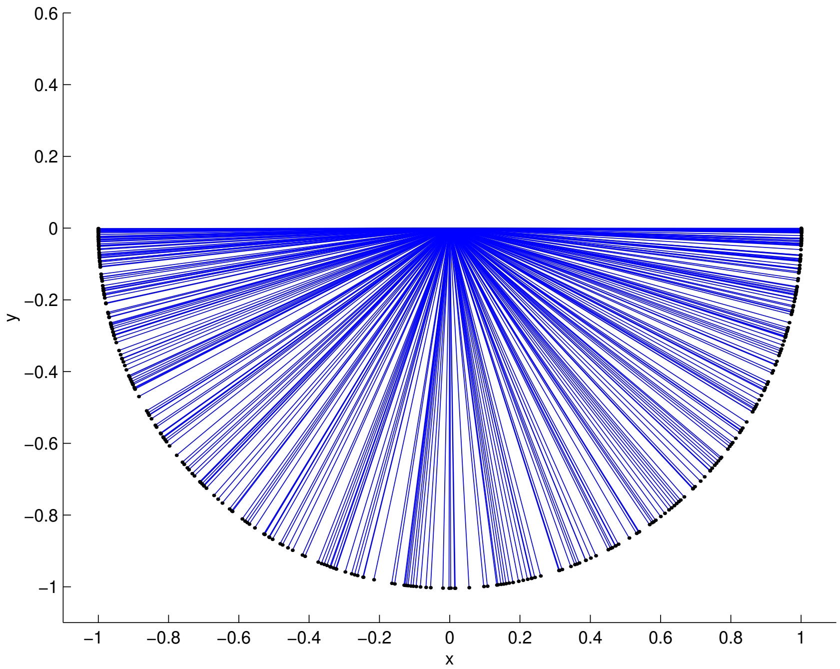

Figure 6: Path of the pendulum using PSRM, $t_0 = 0, t_f = 20, m = 1, h = .01$

8 Step-size Control

8.1 Embedded Runge-Kutta

To implement step-size control, we need some estimate of the local error at the end of each step and then, using that estimate, decide if the current approximation for y_1 is acceptable. Basically, we need to solve each step with two Runge-Kutta methods of different order to obtain two approximations y_1 and $\hat{y}_1$, and thus, an estimate of the error in the less precise approximation, namely $y_1 - \hat{y}_1$. However, this is wasteful as we are essentially solving the problem twice, unless the two methods share information in a way thus doesn't require any addition function evaluations.

In particular, consider two Runge-Kutta methods that have the same stages:

$$
\begin{array}{c|ccccc}
0 & & & & & \\
c_2 & a_{21} & & & & \\
c_3 & a_{31} & a_{32} & & & \\
\vdots & \vdots & \vdots & \ddots & & \\
c_5 & a_{s1} & a_{s2} & \cdots & a_{s(s-1)} & \\
\hline
 & b_1 & b_1 & \cdots & b_{s-1} & b_s \\
 & \hat{b}_1 & \hat{b}_1 & \cdots & \hat{b}_{s-1} & \hat{b}_s
\end{array}
$$

such that

$$y_1 = y_0 + h \sum_{i=1}^{s} b_i K_i \quad \text{and} \quad \hat{y}_1 = y_0 + h \sum_{i=1}^{s} \hat{b}_i K_i.$$

Assume that the orders of the errors of the two approximations differ by 1. That is, if e_1 and $\hat{e}_1$ are the errors in y_1 and $\hat{y}_1$, respectively, then $e_1 = \mathcal{O}(h^p)$, $\hat{e}_1 = \mathcal{O}(h^{\hat{p}})$, and $\hat{p} = p \pm 1$.

These two Runge-Kutta methods are collectively known as an *embedded Runge-Kutta method.*

8.2 Automatic Step-size Control

For a given h, calculate y_1 and $\hat{y}_1$ from our embedded Runge-Kutta method and thus, as pointed out above, the error in the least precise approximation would be $y_1 - \hat{y}_1$. Suppose y has n components, then for each $i \in \{1, 2, \ldots, n\}$, let

$$\delta_i = \mathrm{ATOL}_i + \max(|y_{0i}|, |y_{1i}|)\mathrm{RTOL}_i.$$

where ATOL_i and RTOL_i are the user-defined absolute and relative tolerances respectively. We want the error $y_1 - \hat{y}_1$ to be such that for each i,

$$|y_{1i} - \hat{y}_{1i}| \leq \delta_i,$$

and thus

$$\frac{|y_{1i} - \hat{y}_{1i}|}{\delta_i} \leq 1.$$

Now we can measure the total error $y_1 - \hat{y}_1$ with

$$\epsilon = \sqrt{\frac{1}{n} \sum_{i=1}^{n} \left(\frac{y_{1i} - \hat{y}_{1i}}{\delta_i} \right)^2}$$

Ideally, $\epsilon = 1$ will correspond to the optimal step-size because for each i, $|y_{1i} - \hat{y}_{1i}| = \delta_i$. Now considering the Taylor expansions of y_1 and $\hat{y}_1$, note that $\epsilon = \mathcal{O}(h^{q+1})$ where $q = \min(p, \hat{p})$. Thus

$$\epsilon \approx ch^{q+1} \qquad \text{for some } c \in \mathbb{R} \tag{10}$$

and, as noted above, the optimal step-size h_{opt} occurs when $\epsilon = 1$ so

$$1 \approx c \left(h_{\text{opt}}\right)^{q+1}. \tag{11}$$

Putting (10) and (11) together, we get

$$h_{\text{opt}} = h \left(\frac{1}{\epsilon} \right)^{\frac{1}{q+1}}. \tag{12}$$

At this point, if $\epsilon \leq 1$ then y_1 is accepted and we use h_{opt} as the step-size for the next step. On the other hand, if $\epsilon > 1$ then y_1 is rejected and we repeat the step using a step-size of h_{opt}.

Note, however, that it is very expensive to reject y_1 and repeat a step. In fact, we would rather take a few steps which are smaller than necessary than have to repeat a step. This leads to the following practical considerations:

1. We usually multiply (12) but a safety factor $\alpha < 1$ to make h_{opt} slightly smaller,

thus increasing the likeliness of accepting the next step. Some common values for α are 0.8, 0.9, $(0.25)^{\frac{1}{q+1}}$, or $(0.38)^{\frac{1}{q+1}}$.

2. We do not want the step-size to change too rapidly so we set minimum and maximum factors, α_{min} and α_{max}, by which h can increase. That is, we define h_{new} as follows,

$$h_{\text{new}} = h \min\left(\alpha_{\text{max}}, \max\left(\alpha_{\text{min}}, \alpha\left(\frac{1}{\epsilon}\right)^{\frac{1}{q+1}}\right)\right) \tag{13}$$

where α is the safety factor from above, and typically $\alpha_{\text{max}} \in [1, 5]$, $\alpha_{\text{min}} = \frac{1}{\alpha_{\text{max}}}$. Also it has been shown to be advisable to use $\alpha_{\text{max}} = 1$ after a step has been rejected.

Thus for the revised test, if $\epsilon \leq 1$ then y_1 is accepted and h_{new} is used as the step-size for the next step. Otherwise, if $\epsilon > 1$, then y_1 is rejected and the step is repeated with a new step-size of h_{new}.

8.3 Initial Step-size[3]

This above method works fine for automatic step-size control however, we still need a way to choose the initial step-size.

As discussed above, the local error for y_1 is $\mathcal{O}(h^{q+1})$ and hence for some $c' \in \mathbb{R}$, the local error is approximately

$$\epsilon = \left(c' y^{(q+1)}(t_0)\right) h^{q+1} y = c' h^{q+1} y^{(q+1)}(t_0).$$

$y^{(q+1)}(t_0)$ is unknown but we can replace it with approximations of the first and second derivatives of the solution.

[3]Consult [2] for more information regarding the method outlined here

In [2], the following algorithm based on Forward Euler is presented:

1. Set $d_0 = \|y_0\|$, $d_1 = \|f(t_0, y_0)\|$, and $\delta_i = \text{ATOL}_i + |y_{0i}|\text{RTOL}_i$. Note that d_0 and d_1 are y and y' respectively.

2. As a first guess, if $d_0 < 10^{-5}$ or $d_1 < 10^{-5}$, set $h_0 = 10^{-6}$. Otherwise, set $h_0 = 0.01\left(\frac{d_0}{d_1}\right)$.

3. Take a step of Forward Euler (i.e. calculate $y_1 = y_0 + h_0 f(t_0, y_0)$) and use the result to compute $f(t_0 + h_0, y_1)$.

4. As estimate of y'', compute

$$d_2 = \frac{1}{h_0}\|f(t_0 + h_0, y_1) - f(t_0, y_0)\|.$$

5. If $\max(d_1, d_2) \leq 10^{-15}$, set

$$h_1 = \max\left(10^{-6}, 10^{-3}h_0\right),$$

otherwise set

$$h_1 = \sqrt[q+1]{\frac{0.01}{\max(d_1, d_2)}}.$$

6. Take the initial step-size to be $h = \min(100h_0, h_1)$.

Following this algorithm requires only two function evaluations, both of which are required for embedded Runge-Kutta so it is a cheap way to avoid choosing a very bad initial step-size.

9 Continuous Runge-Kutta

The classical Runge-Kutta methods provide high-order approximations at the end of each step. This is fine if one is only interested in the behavior of the solution at a discrete number of points, however we often want high-order approximations at points between the end points of each step. That is, we like to know $y(t_0 + \theta h)$ for all $\theta \in [0, 1]$, and if possible, we would like to obtain these approximations with a minimum of additional function evaluations.

Start with the s-stage embedded Runge-Kutta method described by the Butcher array

$$\begin{array}{c|c} c & A \\ \hline & b^T \end{array},$$

which we will eventually extend to an s^* stage method. Now consider the formulas of the form

$$u(\theta) = y_0 + h \sum_{i=1}^{s^*} b_i(\theta) K_i$$

where

$$K_i = f\left(t_0 + c_i h, y_0 + h \sum_{j=1}^{i-1} a_{ij} K_j\right) \qquad i = 1, 2, \ldots, s^*$$

and

$$b_i(\theta) = \sum_{k=1}^{q} b_{ik} \theta^k$$

are polynomials to be determined such that $u(\theta) - y(t_0 + \theta h) = \mathcal{O}(h^q)$.

For example, for classic fourth-order embedded Runge-Kutta (ERK4), with $s^* = s$ and $q = 3$, it can be shown that

$$b_1(\theta) = \theta - \frac{3}{2}\theta^2 + \frac{2}{3}\theta^3$$

$$b_2(\theta) = b_3(\theta) = \theta^2 - \frac{2}{3}\theta^3$$
$$b_4(\theta) = -\frac{1}{2}\theta^2 + \frac{2}{3}\theta^3.$$

Note that this solution is continuous but not continuously differentiable — that is, it is not necessarily "smooth".

The continuous Runge-Kutta method of interest to the PSRM (see section 10) is the Dormand-Prince 5(4) embedded pair, with continuous extension of $\mathcal{O}(h^5)$. The Butcher array for this method is

$$\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & \frac{1}{5} & & & & & & \\
\frac{3}{10} & \frac{3}{40} & \frac{9}{40} & & & & & \\
\frac{4}{5} & \frac{44}{45} & -\frac{56}{15} & \frac{32}{9} & & & & \\
\frac{8}{9} & \frac{19372}{6561} & -\frac{25360}{2187} & \frac{64448}{6561} & -\frac{212}{729} & & & \\
1 & \frac{9017}{3168} & -\frac{35}{33} & \frac{46732}{5247} & \frac{49}{76} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
y_1 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0 \\
\hline
\hat{y}_1 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40}
\end{array},$$

and the $b_i(\theta)$ are

$$b_1 = \theta\left(1 + \theta\left(-\frac{1337}{480} + \theta\left(\frac{1039}{360} + \theta\left(-\frac{1163}{1152}\right)\right)\right)\right)$$
$$b_2 = 0$$
$$b_3 = \frac{100}{3}\theta^2\left(\frac{1054}{9275} + \theta\left(-\frac{4682}{27825} + \theta\left(\frac{379}{5565}\right)\right)\right)$$
$$b_4 = -\frac{5}{2}\theta^2\left(\frac{27}{40} + \theta\left(-\frac{9}{5} + \theta\left(\frac{83}{96}\right)\right)\right)$$
$$b_5 = \frac{18225}{848}\theta^2\left(-\frac{3}{250} + \theta\left(\frac{22}{375} + \theta\left(-\frac{37}{600}\right)\right)\right)$$

$$b_6 = -\frac{22}{7}\theta^2\left(-\frac{3}{10} + \theta\left(\frac{29}{30} + \theta\left(-\frac{17}{24}\right)\right)\right).$$

10 Conclusions and Future Goals

The Sequential Regularization and Predicted Sequential Regularization Methods represent a significant improvement over a method such as repeated differentiation in the solution of differential-algebraic equations. Repeated differentiation is unstable even on a stable problem, whereas SRM and PSRM are stable if the problem they describe is stable.

A `fortran` implementation of the PSRM exists and was used to generate the output in section 7. As noted above, this implementation uses Heun's Method (a 2-stage explicit Runge-Kutta method) and uses a constant step-size (specified by the user as a parameter). I plan to implement the PSRM in `matlab` and add automatic step-size control as discussed in section 8. My implementation will most likely make use of the Dormand-Prince 5(4) embedded pair with $\mathcal{O}(h^5)$ continuous extension.

List of Figures

Bibliography

[1] J. Baumgarte, *Stabilization of constraints and integrals of motion in dynamical systems*, Comp. Math. Appl. Mech. Eng. 1 (1972), 1–16.

[2] Gladwell, Shampine, Brankin *Automatic selection of initial step size for an ODE solver* J. Comp. Appl. Math vol 18 (1987), 175–192

[3] U. Ascher and P. Lin, *Sequential regularization methods for higher index DAEs with constraint singularities: Linear index-2 case*, SIAM J. Numer. Anal. 33 (1996), 1921–1940.

[4] U. Ascher and P. Lin, *Sequential regularization methods for nonlinear higher index differential-algebraic equations*, SIAM J. Sci. Comput. 18 (1997), 160–181.

[5] M. Hanke, *Regularization of differential -algebraic equations revisited*, Preprint Nr. 92-19, Humboldt- Univ. Berlin, Fachbereich Mathematik, 1992.

[6] L. Kalachev and R. O'Malley, Jr., *The regularization of linear differential-algebraic equations*, SIAM J. Math. Appl., 1996.

[7] L. Kalachev and R. O'Malley, Jr., *Boundary value problems for differential-algebraic equations*, Num. Func. Anal. Appl. 16 (1995), 363–378.

[8] P. Lotstedt, *On a penalty function method for the simulation of mechanical systems subject to constraints*, Royal Inst. of Tech. TRITA-NA-7919, Stockholm, 1979.

[9] K. Park and J. Chiou, *Stabilization of computational procedures for constrained dynamical systems*, J. of Guidance 11 (1988), 365–370.