Table of Contents

[I.Introduction 2](#_Toc1)

[II.Classic numerical integration methods of Ordinary differential equations 2](#_Toc2)

[1.Single-step integration methods 2](#_Toc3)

[A.Explicit Runge-kutta (RK) 2](#_Toc4)

[B.Implicit Runge-Kutta 3](#_Toc5)

[C. Error and stability 3](#_Toc6)

[2.Multi-step integration methods 3](#_Toc7)

[A.Explicit methods 4](#_Toc8)

[B.Implicit methods 4](#_Toc9)

[C. Error and stability 4](#_Toc10)

[III.Stiffness 4](#_Toc11)

[IV.Differential algebraic equations 5](#_Toc12)

[1. Solvability techniques 5](#_Toc13)

[2. DAE Solvers 6](#_Toc14)

[V.Related topics 6](#_Toc15)

[1.Discontinuous systems 6](#_Toc16)

[2.Real time simulation 7](#_Toc17)

[3.Discrete event simulation 7](#_Toc18)

[VI.Quantization based integration 8](#_Toc19)

[1.Quantized system solver family 8](#_Toc20)

[A.introduction 8](#_Toc21)

[B.General stand-alone QSS solver structure 8](#_Toc22)

[C.General stand-alone QSS solver algorithm 8](#_Toc23)

[D.QSS algorithms comparison 9](#_Toc24)

[i.QSS1: 9](#_Toc25)

[ii.QSS2: 9](#_Toc26)

[iii.LiQSS: 9](#_Toc27)

[iv.Modified LiQSS 10](#_Toc28)

[2.Error and stability 10](#_Toc29)

[3. Differential Algebraic equations 10](#_Toc30)

[4.Mixed-mode method 10](#_Toc31)

# Table of figures

[Figure 1 Stand-Alone QSS structure (Joaquin Fernandez 2014) 9](#_Toc1)

# I.Introduction

Many real-world systems, whether a nuclear plant or a flight training system, exhibit problems that can be very difficult to deal with. Receiving a real training on a helicopter or conducting a real testing on a nuclear plant are very costly in money and time as well as dangerous. Simulation offers a solution to these problems. However, very often these systems are too complex to mimic analytically. Classic numerical solvers offer a solution to this problem, but their results can be unacceptable in large scale systems especially in real time simulation where the simulation time has to race against real time. Recently, new quantization-based techniques have emerged and have shown promising outcomes

# II.Classic numerical integration methods of ODE

Numerical integration methods tackle problems that can be represented by: Where f is a mathematical function in the space , x is a vector of state variables, and t is an independent variable. A special case is an ordinary differential equation which is represented by . These methods share the property of discretizing the time axis such as the famous family of the Runge-Kutta methods that advance in time in a single step and encompass Radau, lobatto and many others. Then, there is the multistep family methods such as the Adams’method and the Backward Differentiation formulae [1], [2]. The discretization may be equidistant (fixed-step algorithms) or dependent on the numerical properties of the differential equations to be solved (variable-step algorithms). The approximation order may be either fixed (fixed order algorithms) or time-dependent (variable-order algorithms) [15].

## 1.Single-step integration methods

### A.Explicit Runge-kutta (RK)

The solution to the differential equation is approximated by fitting an interpolation polynomial through past state and state derivative values. The interpolation polynomial is then used to extrapolate in time to the next sampling instant.

Euler in 1768 provided a numerical method to solve , but it was shown that its global error behaves like Ch where C is a constant and h is the maximal step size. Therefore, Runge in 1895 made new methods that reduce the gloabl error. They are in the form: Where .

Butcher in 1964 somblized the RK methods by the tableau:

The choice of these parameters encapsulates many known antecedent methods such as Forward Euler, Midpoint method, and Heun’s method. It also gave birth to many new sub-RK methods of higher order such as Dormand-Prince 5(4), Verner 6(5), and Fehlberg7(8) with orders 5, 6, and 7 respectively.

### B.Implicit Runge-Kutta

Implicit algorithms avoid the extrapolation by making use of the unknown state derivative at the next sampling instant in the construction of the interpolation algorithm.

They are in the form: Where .

The correspondent Butcher’s tableau is :

Similar to explicit RK methods, the choice of the parameters yields special methods such as the backward Euler, implicit midpoint rule, Lobatto and Radau methods.

### C.Error and stability

The local error for this family is bounded by c whereas the global error is bounded by c where C is a constant, h is the step size and p is the order of the method. To further enhance this method, practical error estimation, higher order extrapolation, and automatic step size control were devised. For instance, the step size is adjusted automatically to make the error does not exceed Atol+yi Rtol where Atol is the absolute tolerance and Rtol is the relative tolerance given the user. In addition, the algorithm designer needs to worry about issues of numerical stability. This latter is defined by the stability function which is in the form: . A method is stable at S={z\ ||F(z)||<1}. it is called A-stable if S contains the left half plane, and it is called F-stable of S is exactly the left half plane. The following table summarizes the stability of some known methods.

|  |  |  |
| --- | --- | --- |
| method | Stability function | Stability region |
| Forward Euler | 1+z | inside the unit circle centered at -1 |
| Backward Euler | 1/(1-z) | Outside the unit circle centered at 1 |
| Trapezoidal | (1+z/2)/(1-z/2) | Left half plane |
| Explicit RK | 1+z+ | Inside an ellipse-like shape left of the imaginary axis |
| Implicit RK | 1+z | Outside an ellipse-like shape right of the imaginary axis |

## 2.Multi-step integration methods

In contrast with single step methods that need only an initial condition x0, multi-step methods need approximation of the solution at k-1 previous points (x0,x1...) in order to compute the solution at step k. That is, a multi-step method aims to make use of its calculation history.

### A.Explicit methods

Similarly to Runge-Kutta methods, multi-step methods,via the choice of the coefficients, encapsulate antecedent methods as special case methods as well as presenting new methods such as the Adams’ and the Nystrom’s methods. They are in the form:

.

### B.Implicit methods

In pursuit of a more accurate approximation, the bs coefficient is kept different than zero to obtain an implicit method that is the form: . Examples are the implicit Adams and the backward differentiation formulas (BDF).

### C.Error and stability

The local error of multistep methods is defined by the term , and it is said consisten of order p if L=O(). Stability is defined by the roots of . The roots must lie on or within the unit circle, and they must be simple if they are on the unit circle, and this is called zero stability (also known as D-stability). Finally, a method is said convergent if it is consistent and stable.

# III.Stiffness

The idea of choosing the best method to use based on the specificity of the problem goes back to when the term stiffness was coined. That is, If we can detect stiffness, we can use the appropriate methods and go back to simple fast explicit methods otherwise. Many informal definitions were given but they all describe the phenomenon by its effects. Brugnano et al. state that a complete formality is of little value to the scientist to solve a problem, and all they need is an operative definition [11]. some definitions linked stiffness to the stability region, and others relied on eigenvalues and conditioning parameters.

First, comparing two error estimates can detect stiffness. Let , so Where R is the stability function. Similarly for . Then is called the error estimate. If we find another error estimate using a low order, and compare it to E. If which is a normal behavior, then the problem is a non-stiff region. If , then hJ is getting near the border which indicates that the problem is getting stiff now.

Second, by detecting when an Eigenvalue lies on the stability border, stiffness is detected. That is, when we have to monitor when . Hence, the challenge to this method is find the largest eigenvalue of the jacobian which is normally accomplished through the Arnoldi algorithm. Another estimation technique of the largest eigenvalue uses the fact which is the Lipschitz constant that can be approximated in methods using Celier et al. mentioned that it was treacherous to rely solely on eigenvalues of the jacobian to conclude anything about stiffness. In fact, for non-linear or time variant systems, eigenvalues can be perfectly tame while the system is extremely stiff. For example, the following system . has -1 and -2 as eigenvalues.

Third, some algorithms check if there is some number of steps rejected, and decide that the region is stiff. However, it is not a reliable technique.

Fourth, another definition linked stiffness to two conditioning parameters [10].

Let the equation . If is a solution to this equation after perturbing the initial condition +, then let the difference –y equals z(t). We then introduce:

If the problem is stiff in [

Finally, it is worth mentioning that once stiffness is detected, all algorithms switch to implicit integration methods.

# IV.Differential algebraic equations

Most the real world problem are modeled as differential algebraic equations (DAE), and solving them is more challenging than solving ordinary differential equations (ODE) or algebraic equations (AE). Luckily, several methods used to tackle ordinary differential equations could be used to tackle these equations after applying other intermediate techniques. For instance, the fully implicit form could be converted to a semi-explicit form: (ODE) and G(y,z)=0 (AE) by setting . y is the differential variable and z is the algebraic variable.

## 1.Solvability techniques

It should be noted that defining the initial conditions is not straight forward as they have to satisfy G. One possible approach is to solve F using known methods, and at every step we solve G for z in function of y. Another approach is to convert F to a difference equation and solve the resulting equations simultaneously. This would lead to a matrix inversion that could be singular. In both approaches, an algebraic equation or a system of AE has to be solved at every step which is computationally expensive. If both approaches do not yield a solution, then the problem is called a higher index DAE.

In the process of reducing a DAE into a system of ODEs and AEs, The Pantelides’ algorithm reduces the index of the problem by eliminating the constraint equation by differentiating it (G) in the hope to get . However, there exists another difficulty when the new system consists of *acausal* equations with loops. The Tarjan’s algorithm is famous for the causalization of the equations via structure digraphs. The tearing algorithm aims to eliminate the algebraic loops. It is worth mentioning that the appropriate choice of the state variables by the modeler is crucial to lessen these aforementioned difficulties. In Fact, Dymola, a modeling and simulation environment, uses these techniques to tackle DAE and it toggles between state variables at run time, and it is able to convert tens of thousands of DAE in ODE in seconds.

## 2.DAE Solvers

DASSL is one of the successful index1-DAE solvers that uses BDF. It is variable step, variable order and it is the default solver of Dymola. However, because of a small step size presented a problem to DASSL, Dymola chose to convert all DAE to ODE before using it. While DASSL separates the engine equation (method equation) from the model equation, another technique called inline integration applies the method equation on the integrator equations of the model, and then it uses the techniques mentioned in the previous section. However, this method too, without a powerful compiler like Dymola, can not tackle large problems because it relies on some hard-coding by the user.

# V.Related topics

## 1.Discontinuous systems

Normal integration algorithms experience algorithms as a sudden a change or appearance of an eigenvalue very far in the left plane, and they react by decreasing the step size. This behavior has two major drawbacks. First, as the step size decreases too much, higher order taylor terms approach zero which makes the method a first order one. Second, a problem with frequent discontinuities would make the step size always small as this latter does not get back to normal quickly.

A time event discontinuity is when the user knows when a discontinuity would happen. That information is given to the integrator algorithm, and this latter decreases the step size at the vicinity, stops the simulation at the given time, and implements a code given after the discontinuity.

However, most of the times it is not know when a discontinuity would happen. Only the conditions at which a discontinuity would happen are known, and this is known as state event discontinuity. In this case, an algorithm monitors when the zero crossing happens, and starts an iteration or interpolation to determine the exact time of the discontinuity. Next, it neglects updates after that time and moves the simulation to the exact time. There are several interpolation and iteration techniques to find the discontinuity time. Newton iteration is a famous one that quadratically converges but it may diverge. The golden section converges linearly. Another approach is to construct a third order interpolation polynomial and find its roots. This approach is guaranteed () conversion. A better approach is to construct the inverse cubic polynomial to avoid solving for the roots.

It is possible that a state variable could reach the border and goes back to its previous operating region. That is, the zero crossing function could be triggered but the state goes back. This behavior could be a practical problem since the algorithm does not know which initial conditions to pick after the zero crossing.

## 2.Real time simulation

The main problem of a real time simulation is the race against real time. This could be achieved by increasing the step size, optimizing the integration algorithm, or using a faster computer. Generally, a low order explicit method is suitable for real time simulations, but due to stiffness a semi-implicit method is usually used instead as explicit methods tend to decrease the step size. Matrix inversion is expensive but it can be predicted unlike iterations, and its computation could be done once in while. There a modified Newton iteration that approximates the Hessian by a diagonal matrix which results in more less expensive steps. This could be better computing the Jacobian once in while could effect stability. However, for large stiff systems, matrix inversions and iterations are a killer in real time simulation

Another approach is to split the model into fast and slow components, and apply different step sizes (multi-rate) and different methods (multi-mode). For example, we could inline slow components with explicit Euler and slow ones with implicit Euler.

Explicit Euler for slow state variable :

Implicit Euler for fast state variable :

For discontinuity, interpolation techniques or regular falsi should be used. A challenge to this approach is to find the slow and fast components. Scheila proposed to use linearization and eigenvalue analysis to accomplish that, but eigenvalues could be evenly spread out or dependent of time.

## 3.Discrete event simulation

All aforementioned methods, despite their difference, extrapolate the next value by discretizing the time axis. A problem carried by time discretization in the simulation of continuous systems is related to the loss of the simulation controlbetween successive discrete instants. A new approach discretizes the state space instead and it is called discrete event simulation (DES). This approach finds the time of the next state value which is considered as an event, and advances simulation time to it. That is, simulation time moves to the next scheduled time of an event.

There are several implementations to this approach. First, a method called Event Graph decomposes the problem into components of highly depended events. It next defines the rules by which each state variable changes value by state transition functions, and then it uses a listener pattern to communicate between the components [16]. However, this method is limited to small systems.

A second method models the problem by a quantized system through a simulation engine called Discrete event system specification (DEVS). This formalism processes an input event trajectory and provokes and output event trajectory. Its representation defines the system by enumerating its state variables and defining the rules as functions. A DEVS *atomic* model is defined by:

*M* = (*X, Y, S, δ*int*, δ*ext*, λ, ta*)

*X* is the set of input event values, *Y* is the set of output event values. *S* is the set of state values. ta(s) : S → R+0 is the time advance function . δint : S → S is the internal transition function. λ : S → Y is the output function. δext : S \* R0+  \* X → S is the external transition function.

Coupled DEVS can be implemented in hierarchical structure of simulators and coordinators. However, it was inefficient because there was too much overhead of synchronization messages which wasted computational effort and time. A similar method was the stand-alone quantized system solver (QSS) and it was implemented as a library in C [5].

# VI.Quantization based integration

## 1.Quantized system solver family

### A.introduction

QSS methods are explicit and linearly-implicit algorithms that aim to replace the classic time slicing methods by a quantization of the states leading to an asynchronous discrete-event simulation model instead of a discrete time difference equation model. The general form of the QSS aproach at solving  where  *Rn* is the state vector,  *Rm* is the input vector, and *t* denotes time is the following:

;  ; = ; q is called the quantized state.

All state derivatives remain constant if the states or inputs do not cross the next threshold, and the time of the crossing can be computed explicitly. QSS algorithms are asynchronous because The time, at which a state variable reaches its next threshold is different for separate states. For instance, state variables with large gradients will get to their thresholds more frequently than states with small gradients. If a state or input gets to its next threshold, a discrete event is triggered, and information is passed on to other integrators that depend on it.

### B.General stand-alone QSS solver structure

The stand-alone QSS software contains three components. First, the Model prepares the structure of the problem and computes the derivatives based on the ODE right hand side function and the quantized state received from the intergrator. Second, the integrator computes the state variable from the structure and the derivative. Finally, the quantizer updates the quantized state variable using the state variable received from the integrator. Figure 1 shows the basic interaction scheme.

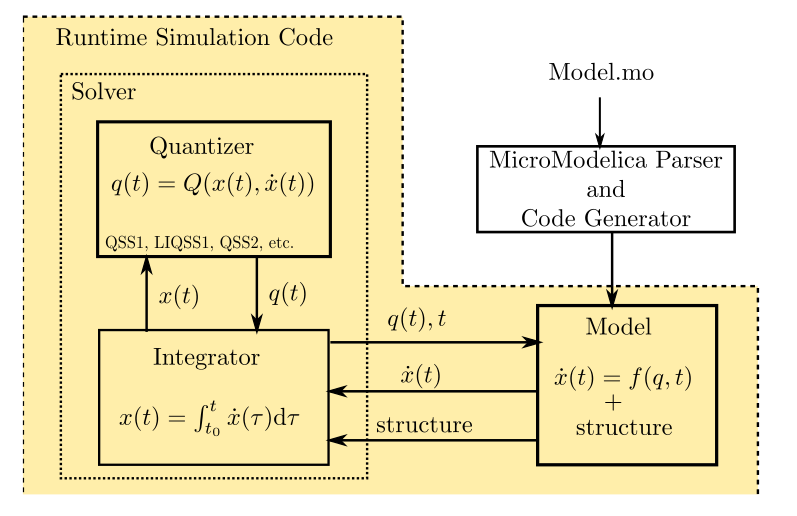


Figure 1 Stand-Alone QSS structure (Joaquin Fernandez 2014)

### C.General stand-alone QSS solver algorithm

Di Pietro et.al use the following algorithm for all QSS solvers:

Begin

While time t is less than the final time

1.Find next minimum time of change and advance the simulation

If event is a state change:

2.Compute the state variable x using Taylor expansion.

3.Update the quantized state q

4.Compute the next time this quantized state should be updated

5.For each other state derivative j that depends on this quantized state

6..Compute the state variable

7..Compute the state derivative

8..Recompute the next time this quantized state should be updated

9..Keep track of this time of change

10.For each zero crossing function j that depends on this quantized state

11. Evaluate (t)

12. Compute next time

If event is a zero crossing event

13.Update the quantized state q

14.Execute the event handler Hi (a variable D will be changed)

For each other state derivative j that depends on the discrete variable D

6.7.8.9

For each zero crossing function j that depends on the discrete variable D

11.12

END

All algorithms of the QSS family differ in computing the state variable x, updating the quantized state q, and computing the next time. In the next section, we show this difference.

### D.QSS algorithms comparison

#### i.QSS1:

;

Compute next:

Recompute next:

QSS1 is not accurate unless it decreases which increases the number of steps. Therefore, versions of higher orders were constructed.

#### ii.QSS2:

;

Compute next:

Recompute next: . This leads to finding the roots of 2 second order polynomials.

Despite their advantages in handling discontinuities, explicit QSS methods are inefficient in solving stiff systems where they produce high frequency oscillations.

#### iii.LiQSS1:

In order to deal with stiffness, LiQSS evaluates the state derivative at future instants using a future quantized state value, as in classic implicit algorithms. The quantized state becomes a future value of the state and it is calculated “smarter”. That is, when we sense a change of the derivative, we neither add nor subtract the quantum , we set the derivative equals to zero. The update in line 3 goes as follows:

;

Where is a linear approximation ; ; ;

is the an entry on the main diagonal of the Jacobian.

is an affine coefficient.

Compute next:

Recompute next:

However, LiQSS also produce fast oscillations in systems where the stiffness is not due to large entries on the main diagonal of the Jacobian matrix.

#### iv.Modified LiQSS1

LiQSS1 can not ensure that is always the future value of because can change its sign due to a change in some other quantized variable . In mLIQSS we check if a change in changes the sign of and a change in changes the sign of then we apply a simultaneous change of the quantized state variables. This simultaneous change is accomplished via two other modifications. First, q is updated as follows: . Second, the linear approximation should no longer equals . and . Therefore, we obtain:

Where and are found using the previous equation and and respectively. Joaquin et al. choose , check if it verifies the previous constraints. If it does not, they choose , check if it verifies the constraints. If it does not, they reduce using a linear approximation. The update of the general aforementioned algorithm, after line 5, goes as follows:

* If
* If perform a simultaneous update of the quantized states.
* For each other state derivative k that depends on , perform lines 6-9
* For each other state derivative j that depends on , perform lines 6-9
* Update the jacobian entries and affine coefficients as needed.

#### v.Modified LiQSS2

The next order modified LiQSS differs from the previous one in the way it computes the state variables and updates the quantized states to account for the second Taylor term. In addition, it checks it there is a significant change in the state first and second derivatives.

## 2.Error and stability

Let then = . Therefore, this is a perturbed representation of the original system with the perturbation is bounded by .

Kofman showed that if f is Lipschitz in x and piecewise continuous in t, then the approximate solution goes to the exact solutionV when the quantum Q goes to zero. Therefore, QSS methods are convergent under these conditions. He showed also that for a stable analytical solution around a point, the usage of a quantum small enough ensures that the approximate solution finishes inside an arbitrary small region around that point. Finally, he computed the global error bound for a linear time invariant as .

Where is the Jordan canonical decomposition of A.

## 3. Differential Algebraic equations

Article 2003

## 4.Mixed-mode method

Article 2014

## 5.Sparse symbolic jacobian computation

Article 2021

**References:**

[1] E. Hairer, S. Nørsett, and G. Wanner. Solving Ordinary Dfferential Equations I. Nonstiff Problems. Springer, Berlin, 2nd edition, 1993.

[2] E. Hairer and G. Wanner. Solving Ordinary Differential Equations II. Stiff and Differential-Algebraic Problems. Springer, Berlin, 1991.

[3] F.E. Cellier and E. Kofman. Continous System Simulation. Springer, New York, 2006.

[4] G. Migoni, E. Kofman, F. Cellier, Quantization-based new integration methods for stiff ODEs., Simulation: Transactions of the Society for Modeling and Simulation International 88 (2012) 387–407.

[5] J. Fern ́andez and E. Kofman, “A Stand-Alone Quantized State System Solver for Continuous System Simulation.,” Simulation: Transactions of the Society for Modeling and Simulation International, vol. 90, no. 7, pp. 782–799, 2014.

[6] Di Pietro, Franco, Gustavo Migoni, and Ernesto Kofman. "Improving linearly implicit quantized state system methods." Simulation 95.2 (2019): 127-144.

[7] Di Pietro, Franco, et al. "Mixed-mode state–time discretization in ODE numerical integration." Journal of Computational and Applied Mathematics 377 (2020): 112911.

[8] S. D. Cohen, A. C. Hindmarsh, P. F. Dubois, Cvode, a stiff/nonstiff ode solver in c, Computers in physics 10 (2) (1996) 138–143.

[9] Ekeland, Kersti, Brynjulf Owren, and Eivorines. "Stiffness detection and estimation of dominant spectrum with explicit Runge-Kutta methods." ACM Transactions on Mathematical Software (TOMS) 24.4 (1998): 368-382.

[10] Mazzia, Francesca, and A. M. Nagy. "Stiffness detection strategy for explicit Runge Kutta methods." AIP Conference Proceedings. Vol. 1281. No. 1. American Institute of Physics, 2010.

[11] Brugnano, Luigi, Francesca Mazzia, and Donato Trigiante. "Fifty years of stiffness." Recent advances in computational and applied mathematics. Springer, Dordrecht, 2011. 1-21.

[12] Kofman, Ernesto, Joaquín Fernández, and Denise Marzorati. "Compact sparse symbolic Jacobian computation in large systems of ODEs." Applied Mathematics and Computation 403 (2021): 126181.

[13] Rocha, Rodrigo Caetano, and Bhalchandra D. Thatte. "Distributed cycle detection in large-scale sparse graphs." Proceedings of Simpósio Brasileiro de Pesquisa Operacional (SBPO’15) (2015): 1-11.

[14] Bender, Michael A., et al. "A new approach to incremental cycle detection and related problems." ACM Transactions on Algorithms (TALG) 12.2 (2015): 1-22.

[15] qss simulation 2008

[16] Buss, Arnold. "Discrete event simulation modeling." Naval Postgraduate School, Monterey, CA (2011).