Runge Kutta integrators for fast and accurate solutions in GEMPACK

Florian Schiffmann

April 14, 2022

Abstract

In GEMPACK, models are always solved as initial value problems(IVP) using the linearized form of the levels equations. While this allows the user to solve each step of the IVP efficiently, the overall accuracy and speed is determined by the integration scheme and the number of integration steps. Up to GEMPACK 12.1, only the Euler, leapfrog midpoint and Gragg's method were available as well as their 2 and 3 point Richardson extrapolations. While Euler provides excellent stability it is very costly to obtain accurate solutions. In contrast the latter two integrators allow for faster convergence but oftentimes suffer from instabilities. Richardson extrapolation for these methods allows for accelerated convergence, however it is not obvious how to choose the parameters and much more costly. In the current beta version of GEMPACK we address this issue by introducing explicit and embedded Runge Kutta (RK) integrators as an alternative.

Our focus in this work is on the embedded RK methods. Using the embedded RK methods we developed a new adaptive step size algorithm that is designed to overcome problem common to CGE models. Such problems include asymptotes in the levels variables as well as coping withe the different scales on which the results can vary. Our algorithm provides rapid convergence towards the true solution as well as increased robustness exceeding that of Eulers method. In addition, the new algorithm allows us to provide users with a component-by-component global error estimate. In all our tests we have found that the error estimates appeared to be upper bounds of the true error. This allows for more confidence in the simulation results without lengthy tests to confirm convergence. Furthermore, this compenent-by-component error estimates are an excellent debugging tool when developing or extending a CGE model.

Due to the dual nature of the objective, i.e. time to solution and accuracy, it is hard to quantify exact benefit from the new methods. However, in all but the simplest test cases, we have found that using adaptive step size embedded RK methods provided solutions at least one order of magnitude closer to the true solution in less than half the time to solution required by the old integration schemes.

1 The CGE model as initial Value problem

A CGE model in levels form with m equations and n variables can be written as

$$\begin{pmatrix} F_1(\mathbf{X},\mathbf{Y}) \\ F_2(\mathbf{X},\mathbf{Y}) \\ \vdots \\ F_m(\mathbf{X},\mathbf{Y}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

where F_i are nonlinear functions of set of m endogenous variable X (unknown) and n exogenous variables Y (known). For any valid set of exogenous variables $\mathbf{Y}$ (i.e. a solution to the nonlinear system exists) the corresponding set of endogenous variables $\mathbf{X}$ can be computed. If an initial solution ($\mathbf{X^0}$, $\mathbf{Y^0}$) to the levels equations is already known, e.g. from economic data, the problem can be reformulated as follows. First we express the change in the exogenous variables $\mathbf{Y}$ as a function of a variable t.

$$\mathbf{Y} = \mathbf{Y^0} + t\Delta\mathbf{Y}$$

inserting this into to original equation and using the chain rule to derive with respect to the new variable t we obtain.

$$\mathbf{J_F}(\mathbf{X})\frac{d\mathbf{X}}{dt} + \mathbf{J_F}(\mathbf{Y})\frac{d\mathbf{Y}}{dt} = 0 \quad or \quad \frac{d\mathbf{X}}{dt} = -\mathbf{J_F}(\mathbf{X})^{-1}\mathbf{J_F}(\mathbf{Y})\frac{d\mathbf{Y}}{dt}$$

with

$$\mathbf{J_F}(\mathbf{Y}) = \begin{pmatrix} \frac{\partial F_1(\mathbf{X},\mathbf{Y})}{\partial Y_1} & \frac{\partial F_1(\mathbf{X},\mathbf{Y})}{\partial Y_2} & \dots & \frac{\partial F_1(\mathbf{X},\mathbf{Y})}{\partial Y_n} \\ \frac{\partial F_2(\mathbf{X},\mathbf{Y})}{\partial Y_1} & \frac{\partial F_2(\mathbf{X},\mathbf{Y})}{\partial Y_2} & \dots & \frac{\partial F_2(\mathbf{X},\mathbf{Y})}{\partial Y_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial F_m(\mathbf{X},\mathbf{Y})}{\partial Y_1} & \frac{\partial F_m(\mathbf{X},\mathbf{Y})}{\partial Y_2} & \dots & \frac{\partial F_m(\mathbf{X},\mathbf{Y})}{\partial Y_n} \end{pmatrix}$$

being the Jacobian of $\mathbf{F}$ with respect to $\mathbf{Y}$ and $\mathbf{J_F}(\mathbf{X})$ the Jacobian with respect to $\mathbf{X}$ defined in the same way. In the most general case, this defines a system of partial differential equations. While it is impossible to find an analytic solution for this problem, it can be treated as an initial value problem (IVP). This new form simplifies the problem from solving a nonlinear system to solving a linear system and accurately integrating the endogenous variables.

2 Integrators Basics

Before deciding on a specific integrator for a certain problem it is important to understand certain basic concepts related to integrating initial value problems.

2.1 Order of Integrators

Most integrators are classified by their order of the truncation error with respect to the step size h. It is important to note, that if more than a single integration step is performed, there is a difference between the local (end of each step) and global truncation error (error over all steps) For a certain order p, the local truncation error is of order $\mathcal{O}(h^{p+1})$, whereas the global truncation error is only

$\mathcal{O}(h^p)$. There are classes of integrators, for which the global truncation error behaves differently, however all integrators used in GEMPACK follow the above definitions. I.e. haling the step size for a first order method reduces the local error by $\frac{1}{4}$ whereas for a forth order method the local error is reduced by $\frac{1}{32}$. In general, higher order methods are more costly to evaluate, due to the need to evaluate gradients at more than a single position. This slightly offsets the gains from the accelerated convergence, but it nevertheless remains true, that more accurate solutions will be obtained faster.

2.2 Richardson extrapolation

One way to exploit this systematic reduction in error is known as Richardson extrapolation. Assuming we have a p-th order integrator, we can use the knowledge of the convergence of the global truncation error and write the result a step size h_1 as follows:

$$A(h_1) = A_{exact} + C_p h_1^p + C_{p+1} h_1^{p+1} + ... \tag{1}$$

where $A(h_1)$ is the solution obtained by integrating with step size h_1, A_{exact} is the unknown exact solution for the problem and C_x being constant prefactors for all the error of order x. By computing addition solutions with different step sizes we obtain more equations of the form

$$A(h_n) = A_{exact} + C_p h_n^p + C_{p+1} h_n^{p+1} + ... \tag{2}$$

If we now truncate the error expansion at order $p + n - 2$ we obtain a system of n linear equations with n unknown, which can be solved up for the exact solution. Due to the truncation, higher order error terms still remain but the effective order of the integrator increases by $n - 1$. E.g. using Eulers method (1^{st} order) in combination with a 3 solution Richardson extrapolation yields a result that is correct up to 3^{rd} order.
More importantly, it can be shown that for certain methods, e.g. leapfrog midpoint and modified midpoint (Gragg), the integration error only contains even orders. Following the procedure outlined above, only even powers in h have to be accounted for. In this case each additional solution allows us to increase the accuracy by 2 orders instead of one.

2.3 Adaptive step size integration

When integrating an IVP with fixed step size it is impossible to predict the overall error prior to performing the computation. Therefore, it can be convenient to use non constant step sizes that aim to achieve a certain accuracy. These methods are known as adaptive step size methods. For such a method to work and estimate of the accuracy of the integration and the expected convergence of the integrator is required. There are various methods with which an error estimate for the local truncation error can be obtained. One such method exploits results from integrators of different orders. Let us assume we have the result of an integrator with order p (A_p) and $p + 1$ (A_{p+1}). As we expect the higher order solution to more accurate than the lower order solution and the higher order error terms smaller than the leading order error term, we can take

the difference between the two as an upper bound for the true error.

$$\Delta = |A_p - A_{p+1}| \tag{3}$$

With this error estimate and the knowledge of the order local truncation error of our method, we can now compute the required step size scaling (q) such that the error is below a desired threshold ϵ:

$$q \approx \left(\frac{\epsilon}{\Delta}\right)^{\frac{1}{p+1}} \tag{4}$$

Such an error estimate is readily available as part of the Richardson procedure described above. Another set of methods that provide an error estimate are embedded Runge Kutta methods in which different linear combinations of gradient approximations are used to create two set of solutions. The advantage of the latter is the relatively small overhead incurred as many of the gradient approximations are shared between the two solutions. A more detailed description of these methods will be provided below.

2.4 Stability of the Integrator

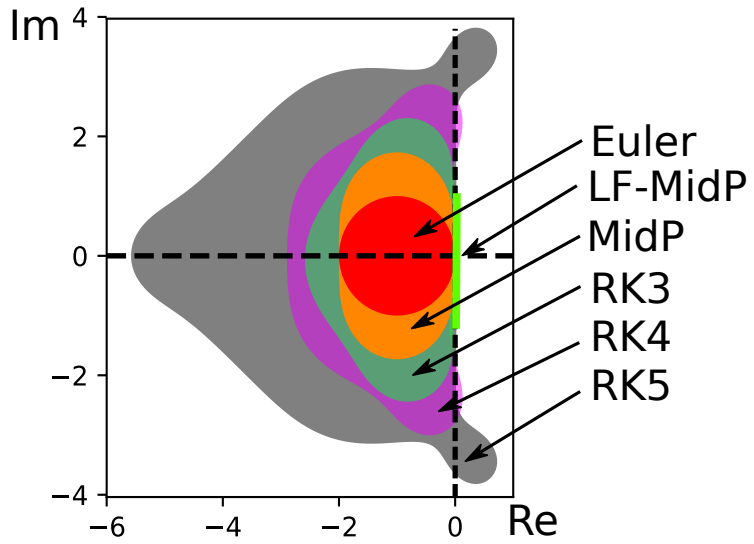
For certain problems an integrator might fail to produce a convergent solution. A reason for this behavior is that the step size is to large, i.e. the step size is outside the stability region for this problem. The worst case scenario here is that the stability region for an integrator for a specific problem is so small, that it is impossible to obtain an accurate integration. These type of problems are known as stiff problems. It is important to note, many problems are only stiff over a integration interval, whilst other regions can be integrated without problems. Whilst there are many attempts to characterize the cause of stiffness, so far it turns out to be more of a phenomenological observation than a precise mathematical theory and one of the definitions is:
An IVP [initial value problem] is stiff in some interval [a,b] if the step size needed to maintain stability of the forward Euler method is much smaller than the step size required to represent the solution accurately. (Ascher, U. M. and Petzold, L. P. (1998), Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations, Philadelphia:SIAM.)
To understand this statement lets assume we integrate over an interval $[0, a]$ with $a > b$. We split this integration into n steps such that $h = \frac{a}{n}$. At the end of each step we compare the result to the true solution to determine the accuracy. If a stiff region is encountered during in one of the integration steps, we might still find that good accuracy is obtained in this integration step even though h is larger than the stability threshold. However, continuing the integration past this point the integration error will start to grow explosively. From this point onwards, it is impossible to recover a bound integration error no matter how small we make the following steps. The system is unstable. This means, even though the critical step was sufficiently accurate, a smaller step size is required to retain stability
A criterium to quantify the stability of different integrators and compare them is the so called A-stability. This measure analyses if and for which step sizes an integrator approaches zeros solution for $t \to \infty$ for the ode

$$\frac{x'}{t} = kx \quad \text{with the Solution} \quad x = e^{kt}$$

for k < 0. For a simple integrator like Eulers method this is straightforward to analyses. For this ode, the general solution using Eulers method can be written as:

$$x_{n+1} = x_n + hkx_n \quad or \quad x_{n+1} = (1+hk)x_n \quad or \quad x_{n+1} = (1+hk)^{n+1}x_0$$

Hence, the method only converges to 0 if $|(1+hk)| < 1$. This indicates, that Eulers method is only conditionally stable for this problem. The same analysis, even though somewhat more involved, can be preformed for other integrators.



As the figure shows, Runge Kutta methods have somewhat larger stability regions than Eulers method. However, it has to be noted that RK methods have much higher computational demands than Euler and the increase in stability is not proportional to the computational cost. Therefore it can be computationally more efficient to use lower order integrators for stiff problems, even though convergence is slower.
For most CGE models we find that semi stiff regions can be found where constituents of CES or CET functions quickly approach 0. Therefore it is important to understand the notion of stiffness in the framework of CGE models as many of the economic functions exhibit properties that can cause the stiffness in various regions of the integration space. If a simulation is performed that causes a steep decline in certain demands the integration can become unstable. In such a case the step size has to be reduced and potentially a more appropriate (lower order) integrator has to be chosen.

3 Integrators Implemented in GEMPACK

3.1 Eulers method

Eulers method is the most basic integrator offered in GEMPACK. It is a first order explicit integration scheme. At any point it simply follows the local gradient

for the step size specified, i.e.

$$x_{n+1} = x_n + h\frac{dx_n}{dt}$$

It is important to note, that Eulers method is the only intergrator, that purely relies on local information, i.e. information purely depending on the current state of the system. For the better or the worse, this makes Eulers method oblivious to non analytic changes in the models.

3.2 Leapfrog midpoint and Graggs method

Both leapfrog midpoint (LF) and Graggs method are second order explicit integrators. Graggs method adds an additional smoothing step to the leapfrog method which greatly increases the accuracy but doe not change the LF inherent properties. Both methods fall in the class of linear multistep methods. This is a special class of integrators Both methods start with a single Euler step:

$$x_1 = x_0 + \frac{h}{n}\frac{dx_0}{dt}$$

for every following step, the current gradient is applied twice to the previous value

$$x_{n+1} = x_{n-1} + \frac{2h}{n}\frac{dx_n}{dt}$$

This can be seen as x_{n-1} jumping over x_n, hence the name leapfrog midpoint. In case of Graggs method, the final step is followed by an averaging of x_{n-1} and x_n and corrected by the gradient at the final position

$$x_{end} = x_{n-1} + x_n + \frac{h}{n}\frac{dx_n}{dt}$$

While having better convergence properties than Euler, both Gragg and LF-midpoint can struggle, if variables in a CGE model approach their asymptotic limit. In this case, it is likely, that stability issues can be observed with these methods. Due to the nature of the integrator, stability issues might not be easily fixed by increasing the number of steps. Instead, it is necessary to restart the integrator. In GEMPACK this is equivalent to using additional subintervals.

3.3 Runge-Kutta methods

3.3.1 Explicit Runge Kutta Methods

Runge Kutta methods employ a different strategy compared to the above methods. Instead of using a computed gradient directly, gradients are evaluated at different sample points and are then combined in a linear combination to advance the system. The exacts details on how to chose the sample points and prefactors are rather involved but the basic idea can be illustrated as follows. Given an initial value problem

$$\frac{dx(t)}{dt} = f(t, x(t)) \quad \text{and} \quad x(t_0) = x_0$$

A first order approximation can be obtained from the left hand rectangle method (left Riemann sum)

$$\int_{t_0}^{t_1} \frac{dx(t)}{dt} dt = x(t_1) - x(t_0) \approx (t_1 - t_0) \frac{dx(t_0)}{dt}$$

Solving for $x(t_1)$ one finds back Euler's method, which can be seen as the first order Runge-Kutta method. It is however possible to use higher order approximations for the integral above, e.g. a second order approximation

$$\int_{t_0}^{t_1} \frac{dx(t)}{dt} dt = x(t_1) - x(t_0) \approx (t_1 - t_0) \frac{\frac{dx(t_0)}{dt} + \frac{dx(t_1)}{dt}}{2}$$

As it is apparent from this formulae, gradients at points t_0 and t_1 need to be calculated. However, this formula is not unique and other ways exist to approximate second and higher orders. This as well as the question how to compute the gradients away from t_0 is formalized in the Runge Kutta methods by using constraints on the error bounds (order conditions). The number of order conditions quickly rises with the order of the integrator and more and more gradient evaluations are required allow for sufficient degrees of freedom to fulfill them. A convenient way to summarize the stages for Runge Kutta methods is in a Butcher Tableau.

$$\begin{array}{c|cccc} 0 & 0 & 0 & ... & 0 \\ c_2 & a_{21} & 0 & ... & 0 \\ c_3 & a_{31} & a_{32} & ... & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ c_s & a_{s1} & a_{s2} & ... & 0 \\ \hline & b_1 & b_2 & ... & b_s \end{array}$$

The rows above the line indicate how the intermediate gradients have to be obtained via

$$\frac{dx(t_0 + hc_i)}{dt} = f\left(t_0 + hc_i, x_0 + h \sum_{j=1}^{i} a_{ij} \frac{dx(t_0 + hc_j)}{dt}\right)$$

And the row below the line gives the recipe on how to combine the gradients to obtain the value at $x(t + h)$ via

$$x(t + h) = h \sum_{i}^{s} b_i \frac{dx(t_0 + hc_i)}{dt}$$

While being a lot more complex than the previous methods, Runge Kutta methods offer a way to obtain higher order convergence at low cost and are therefore the de facto standard in many fields dealing with the integration of IVP. Currently, two explicit Runge Kutta integrators are available in GEMPACK:

- Midpoint method (RK2), 2^{nd} order, 2 stages
- Classic fourth order method (RK4), 4^{th} order, 4 stages

3.3.2 Embedded Runge Kutta Methods

In addition to the standard Runge Kutta methods a second class of Runge Kutta methods is of great importance. Imagine a problem that exposes near linear behavior in certain regions whilst other region are highly non linear. The step size in the nonlinear region will have to be chosen relatively small to obtain an accurate solution (suppression of higher order errors). However, the near linear region permits the use of larger step sizes for obtaining the same accuracy. Using a fixed step size, we are bound by the small step size from the nonlinear region to accurately integrate the problem and thus perform a lot of unnecessary work in the linear region. In the Runge Kutta framework such methods are known as embedded methods. The general idea is to chose the parameters in the Butcher table such that in the end (under the horizontal line) two different methods can be constructed with differing order, e.g. a 5^{th} order method with an embedded 4^{th} order method. The most simple method of this kind is the Heun-Euler method. While it is not of particular importance in practice it easily illustrates the idea. The Butcher tablau for this method looks as follows:

0	0	0	
1	1	0	
	$\frac{1}{2}$	$\frac{1}{2}$	Heuns method
	1	0	Eulers method

When computing Heuns second order method two gradient evaluations are required of which the first is exactly the one needed for Eulers method. This means that in this case we obtain the embedded method as part of our existing second order method without additional cost. Unfortunately, for higher order methods it is more involved to derive embedded schemes and additional gradient evaluations are needed to obtain sufficient accuracy in both the integration and embedded method. This means that generally embedded methods come with an additional cost. However, the efficiency gain from using adaptive step sizes typically outweighs the additional cost. Furthermore, it is possible to collect the integration error and report an accuracy estimate for all variables allowing for an increased confidence in the results. Currently 2 different embedded methods are implemented in GEMPACK:

- Bogacki-Shampine (BoSha32), 3^{rd} order + emdedded 2^{nd} order, 4 stages
- Dormand-Prince (DoPri54), 5^{th} order + emdedded 4^{th} order, 7 stages

4 Practical Notes on Integrators in GEMPACK

4.1 Integrator options in GEMPACK 12.1 and earlier

Fig. 4 gives an overview of the available CMF statements. The options for GEMPACK 12.1 and earlier are listed in the left hand side of the graph. Up until GEMPACK 12.2 only three Euler, LF-Midpoint and Graggs method were available as option for the **method** keyword.
In addition to setting the step size for the integrator, the **steps** keyword acts as a switch between a normal multistep integration, a 2-stage or 3-stage Richardson extrapolation depending on the how many integers are given. I.e. if a single

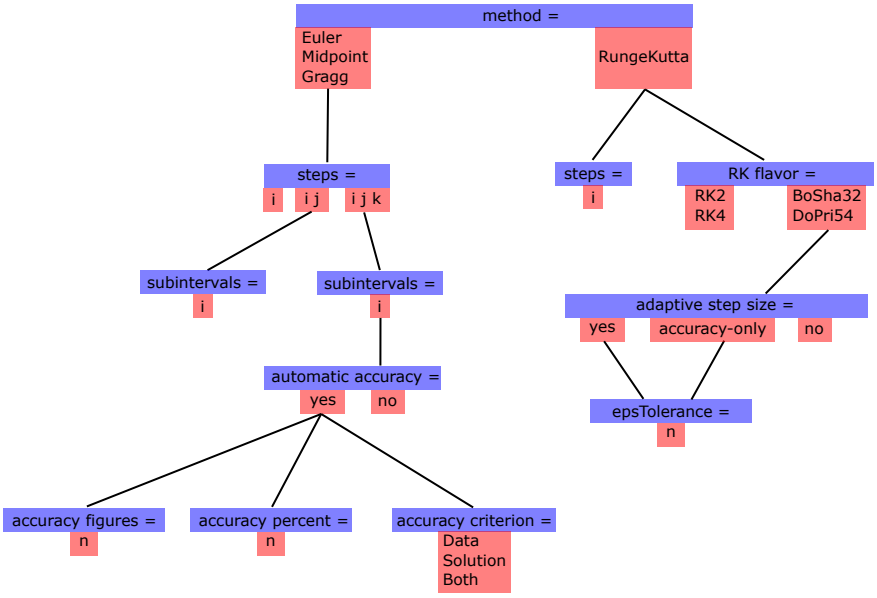


Figure 1: interdependence of GEMPACK CMF options for chosing Integrators

number i is present, a multistep calculation with i steps is performed. Two numbers i, j will trigger a 2 stage Richardson extrapolation using a i-step and j- step multistep integration. Similarly, three numbers i,j and k trigger a 3-stage extrapolation. For Euler the only constraint is, that $i < j$ or $i < j < k$ respectively. For Gragg and LF-midpoint, the additional constraint is that i,j and k are either all even or odd. While odd numbers are permissible in GEMPACK, it is recommended to use even numbers for Gragg and LF-midpoint, as the odd solution are known to more strongly deviate from the true solution.
Both 2-stage and 3=stage extrapolated results can be used in conjunction with **subintervals**. For extrapolated methods, subintervals act as the effective number of steps performed. The easiest way is to think of the extrapolated method as an integrator itself. In this case the number of subintervals specifies the number of steps performed with the new integrator. This directly hints at the usefulness of subintervals. As discussed above, Richardson extrapolation produces a higher order result. Therefor it is more beneficial to perform more steps of the higher order method than increasing the number of lower order steps used in the extrapolation itself.
In case of a 3 stage Richardson extrapolation, it is possible to obtain an accuracy estimate. If the user chooses, a detailed report can be obtained using the accuracy extrapolation file option as detailed in the GEMPACK manual. In addition, if used in combination with subintervals, these accuracy estimate can be used adjust the number of subintervals such that the solution remains within certain the bounds. This option is activated by the **automatic accuracy** keyword. The bounds are specified via the percentage (**accuracy percent**) of results that have to be converged up to a certain number of significant figures (**accuracy figures**). In addition the user can specify whether this criterion

applies to coefficient, Variables or both (**accuracy criterion**).

4.2 Integrator options since GEMPACK 12.2

In GEMPACK 12.2 Runge Kutta integrators were added to the available options. The CMF option can be found in the right side of Fig. 4. As there are many different parametrizations of Runge Kutta type integrators the **method** keyword has to be set to RungeKutta and the type of integrator is specified in the **RK flavor**. For Runge Kutta methods Richardson extrapolation is not available. While possible it is more costly to apply Richardson extrapolation compared to simply using higher order Runge Kutta methods. E.g. 1-2 RK4 Richardson extrapolation requires the analogue of 12 stages and effectively yields a 5^{th} order method with 4^{th} order error estimate, which is identical to the DoPri54 integrator, which only require 7 stages.
In case of the embedded methods BoSha32 and DoPri54 accuracy estimates are available and written to the <sl4>.acc file. Furthermore, it is possible to chose **adaptive step size** integration suing the scheme discussed above. Two options are available for adaptive step size integration. The first is full adaptive step size (adaptive step size = yes). In this case, at each step, the error estimate is evaluated and the step size is adjusted accordingly (both larger or smaller). In addition, range checks and assertion will be set to warning level and steps will be redone with smaller step size (higher accuracy) as soon as such a warning is triggered. This behavior can be important to avoid arithmetic problems resulting from such a failed check in the remainder of the Runge Kutta step. As such a failure will most likely trigger a reevaluation of the step anyway, this behavior is likely to save computation time. The second option "accuracy only" will only react to the accuracy of the step and ignore assertion or range check errors.
The accuracy criterion for adaptive step size is specified as "epsTolerance". For most simulations a value of 0.1 should be sufficient to produce accurate results. For more accurate solutions a value of 0.01 should be sufficient. While there is no limit on epsTolerance, we have found that values below 0.001 are hard to achieve due to numerics. A more detailed discussion of what is measured as accuracy and eps Tolerance will be given below.

4.3 The Error Metric and Error Accumulation

The results in a CGE model can be of very different orders of magnitude. E.g. ordinary changes in values can be in the millions or trillions, while some percent changes can be significant if up to the fourth decimal. For this reason, it is insufficient to purely chosing absolute errors as the significance of errors for large variables would be overestimated. Similarly, relying on relative errors will overestimate the significance of errors for small variables would be overestimated. For this reason, we decide define the error metric as:

$$E = \left| \frac{\Delta}{\max(1, V)} \right| \tag{5}$$

where Δ is the error estimate for the solution of variable V. This means, if a solution V is larger than 1 the relative error is considered otherwise we use the absolute error. The error metric defined in this way reduces the bias at the extremes, i.e. very large and very small values. However, it is important to

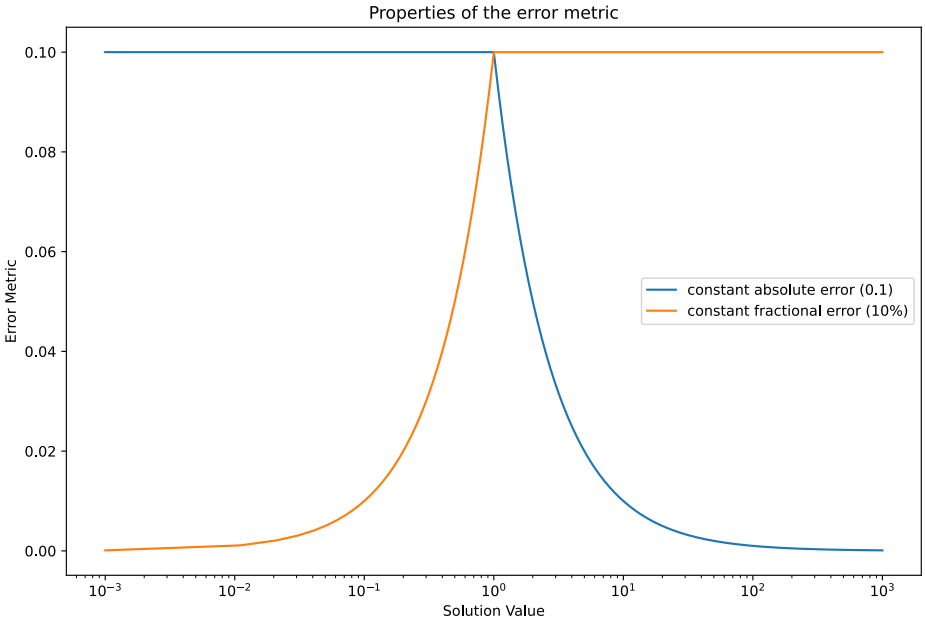


Figure 2: Behavior of the error metric for constant relative error and constant absolute error as function of the solution value V

note, that it creates a bias for values in the proximity of 1 as can be seen in Figure 4.3.
When using embedded methods, an error propagation is performed to estimate the error of the cumulative solution. The error propagation in case of ordinary change variables is trivially derived as:

$$\Delta_i^{Cum} = \Delta_{i-1}^{Cum} + \epsilon_i \tag{6}$$

Where Δ_i^{Cum} is the cumulative error at step i and $\Delta_0^{Cum} = 0$. For percent changes we will have to derive the error propagation from the formula used to accumulate the results. The accumulation formula is given as

$$X_i^{Cum} = X_{i-1}^{Cum} + \left(1 + \frac{X_{i-1}^{Cum}}{100.0}\right) V_i \tag{7}$$

With X_i^{Cum} the cumulative solution at step i and V_i the Solution at step i and $X_0^{Cum} = 0$. Applying Gaussian error propagation to this formula the cumulative error estimate is:

$$\Delta_i^{Cum} = \Delta_{i-1}^{Cum} + \frac{V_i}{100.0}\Delta_{i-1}^{Cum} + \left(1 + \frac{X_{i-1}^{Cum}}{100.0}\right)\epsilon_i \tag{8}$$

At the end of a simulation with embedded methods, the cumulative error estimates are written to the file <sl4>.acc, where <sl4> indicates the basename of the solution file. Furthermore, a face value (quality estimate) is assigned to the simulation according to the value of the error metric for the variable with the worst error metric (Δ^{Max}). This is a simple progression in brackets of size 0.02

starting from $0< \Delta^{Max} < 0.02$ corresponding to a face value of 10 down to a face value of 1 for $\Delta^{Max} > 0.18$.

4.4 Adaptive Step Size Adjustment in GEMPACK

As discussed above, the solution values, and thus the associated errors can vary over a large range in CGE simulations. Hence, using the straightforward approach to step size adjustment

$$q \approx \left(\frac{\epsilon}{\Delta}\right)^{\frac{1}{p}} \tag{9}$$

is it does not account for the different scales. Instead, we base the formula on the error metric:

$$q \approx \left(\frac{\epsilon}{\frac{\Delta}{\max(1,V)}}\right)^{\frac{1}{p}} \tag{10}$$

where ϵ is set via the command file option **epsTolerance**.
As we are aware, we are only working with approximations, using the full scaling might lead to too large step sizes and thus the need to redo steps. To avoid this, we introduce a safeguard by introducing a "chicken factor" of 0.85 to increase the accuracy of the steps taken. E.g. for a 5^th order method this should decrease the error by a factor of $0.85^5 \approx 0.45$. A second safeguard is to limit the size of the step size adjustment. As a truncated error expansion is used in the derivation of the step size adjustment formula, we don not posses sufficient information to predict the effect of higher order terms. These effects become more and more important the larger the step size adjustment is. Hence we limit the amount of the step size adjustment according to $0.5 < q < 2.0$. Therefore, the step size adjustment in GEMPACK is computed according to:

$$q = \max\left[0.5, \min\left[2.0, 0.85\left(\frac{\epsilon \max(1,V)}{\Delta}\right)^{\frac{1}{p}}\right]\right] \tag{11}$$

Due to the safeguards implemented in the step size adjustment, real simulations are typically more accurate than the value ϵ.

4.5 Using the .acc file

In levels models, the accuracy can simply be tested by how well each equation is satisfied. In linearized models, this is a much harder task. The only true test is to run a sequence of simulations with different number of steps until the results remain unchanged. In practice, this is not a viable strategy. For embedded Runge Kutta methods, GEMPACK provides the <sl4>.acc file, which contains the error metric estimates for the cumulative solutions. This file is written in GEMPACK's sol file format, i.e. one header per retained variable in the system. The estimates are given as absolute values, hence sorting in descending order according to the maximum value will give an overview of the worst converged variables. For a well behaved simulations, all values on the .acc file should be very small. If this is not the case there are three different reasons, why large error metrics can be seen.

The first is simply that not enough steps were used and the model is not accurately solved. For such a simulation, it is expected that most variables have sizable errors. The fix is simply to rerun the simulation with more steps for simulation without adaptive step size or a smaller value for **epsTolerance**.
The second case is due to the bias in the error metric. This can be triggered for example, if a variable is derived as the difference between two large ordinary change variables. Due to the way the error metric is computed, the absolute error in the two variables can be non negligible but its relative value is small. When the difference is taken, the resultant error will be the sum of the two absolute errors. In case that the difference between the two variables is small, the error metric will be large. For this problem, there is no adhoc solution as it depends on the meaning of the variable. If only it's relative size matters, the accuracy warning might be safely ignored. Otherwise, if it's absolute value is of interest, a more accurate integration method will have to be chose. The latter can be problematic, as it might be hard to achieve the required numerical accuracy in the original large change variables. On the .acc file, such a problem typically appears as a large error metric in a single component of a variable or two.
The third type of error is related to errors in the model code, such as incorrect linearization or updates. In this case the integrator has a systematic problem integrating a subset of connected variables. Hence, on the .acc file, it can be typically seen as larger errors in a set of economically connected variables. In this case it is important to run additional tests to ensure that the model code is correct.

4.6 Which integrator to use?

For most purposes we recommend using the DoPri54 integrator in conjunction with adaptive step size=yes. In most circumstances a value of epsTolerance=0.1 should give a sufficiently accurate solution. However, a lower value might be required if the problem takes many steps to solve or certain variables need to be highly accurate (e.g. for the computation of second order properties). In the first case, this is because epsTolerance only controls the per step error while cumulative solution is a function of all these errors. Hence, the more steps are used, the larger the cumulative error becomes. With its capability to react to assertion and range check errors, the adaptive step size option can avoid problems of Coefficients exceeding their bounds, which otherwise might only be achieved by using a large number of integration steps. In addition, the availability of the .acc file is a free benefit when using these methods and allow for insight into the quality of the solution.
The drawback is the slightly higher cost incurred by having more stages than the explicit methods. If the user is concerned about time to solution, e.g. in recursive dynamic models, it can be beneficial to only use the adaptive methods in the shocked years and switch to RK4 in the remainder.
For simulations, with large shocks, it can sometimes be more efficient to switch to the BoSh32 integrator. The reason is, that the accuracy is not the main factor causing step recomputations but variables exceeding their defined bounds due to small integration errors. In this case, the lower order intergrator can be more time efficient as the number of steps required for an accurate solution will be similar but the cost per step is reduced.