Dense Output

Lawrence F. Shampine[1] and Laurent O. Jay[2]
[1]Department of Mathematics, Southern Methodist University, Dallas, TX, USA
[2]Department of Mathematics, The University of Iowa, Iowa City, IA, USA

Introduction

We solve numerically an *initial value problem, IVP*, for a first-order system of ordinary differential equations, ODEs. That is, we approximate the solution $y(t)$ of

$$y'(t) = f(t, y(t)), \qquad t_0 \le t \le t_F$$

that has given initial value $y(t_0)$. In the early days this was done with pencil and paper or mechanical calculator. A numerical solution then was a table of values, $y_j \approx y(t_j)$, for mesh points t_j that were generally at an equal spacing or *step size* of h. On reaching t_n where we have an approximation y_n, we *take a step* of size h to form an approximation at $t_{n+1} = t_n + h$. This was commonly done with previously computed approximations and an Adams-Bashforth formula like

$$y_{n+1} = y_n + h\left[\frac{23}{12} f_n - \frac{16}{12} f_{n-1} + \frac{5}{12} f_{n-2}\right]. \quad (1)$$

Here $f_j = f(t_j, y_j) \approx f(t_j, y(t_j)) = y'(t_j)$. The number of times the function $f(t, y)$ is evaluated is an important measure of the cost of the computation. This kind of formula requires only one function evaluation per step.

The approach outlined is an example of a *discrete variable method* [9]. However, even in the earliest computations, there was a need for an approximation to $y(t)$ between mesh points, what is now called a *continuous extension*. A continuous extension on $[t_n, t_{n+1}]$ is a polynomial $P_n(t)$ that approximates $y(t)$ accurately not just at end of the step where $P_n(t_{n+1}) = y_{n+1}$, but throughout the step. Solving IVPs by hand is (very) tedious, so if the approximations were found to be more accurate than required, a bigger step size would be used for efficiency. This was generally done by doubling h so as to reuse previously computed values. Much more troublesome was a step size that was not small enough to resolve the behavior of the solution past t_n. Reducing h to h' amounts to forming a new table of approximations at times $t_n - h', t_n - 2h', \ldots$ and continuing the integration with this new table and step size. This was generally done by halving h so as to reuse some of the previously computed values, but values at $t_n - h/2, t_n - 3h/2, \ldots$ had to be obtained with special formulas. Continuous extensions make this easy because the values are obtained by evaluating polynomials. Indeed, with this tool, there is no real advantage to halving the step size. To solve hard problems, it is necessary to vary the step size, possibly often and possibly by large amounts. In addition, it is necessary to control the size of the step so as to keep the computation stable. Computers made this practical. One important use of continuous extensions is to facilitate variation of step size.

Some applications require approximate solutions at specific points. Before continuous extensions were developed, this was done by adjusting the step size so that these points were mesh points. If the natural step size has to be reduced many times for this reason, we speak of *dense output*. This expense can be avoided with a continuous extension because the step size can be chosen to provide an accurate result efficiently and a polynomial evaluated to obtain as many approximations in the course of a step as needed. This is especially important now that problem-solving environments like MATLAB and graphics calculators are in wide use. In these computing environments, the solutions of IVPs are generally interpreted graphically and correspondingly, we require approximate solutions at enough points to get a smooth graph.

The numerical solution of ODEs underlies continuous simulation. In this context, it is common that a model is valid until an event occurs, at which time the differential equations change. An event is said to occur at time t^* if $g(t^*, y(t^*)) = 0$ for a given event function $g(t, y)$. There may be many event functions associated with an IVP. *Event location* presents many difficulties, but a fundamental one is that in solving the algebraic equations, we must have approximations to $y(t)$ at times t that are not known in advance. With a continuous extension, this can be done effectively by testing $g(t_n, y_n)$ and $g(t_{n+1}, y_{n+1})$ for a change of sign. If this test shows an event in $[t_n, t_{n+1}]$, it is located accurately by solving $g(t^*, P_n(t^*)) = 0$.

In the following sections, we discuss briefly continuous extensions for the most important methods for

solving IVPs numerically. In the course of this discussion, we encounter other applications of continuous extensions. Providing an event location capability for a wide range of methods was the principal reason for developing the MATLAB ODE suite [13]. A few details about this will make concrete our discussion of some approaches to continuous extensions.

Linear Multistep Methods

Adams-Bashforth formulas are derived by approximating the integrated form of the differential equation

$$y(t) = y(t_n) + \int_{t_n}^{t} f(x, y(x))\, dx,$$

with an interpolating polynomial. Previously computed slopes $f_n, f_{n-1}, \ldots, f_{n-k+1}$ are interpolated with a polynomial $Q(x)$ and then

$$P_n(t) = y_n + \int_{t_n}^{t} Q(x)\, dx. \tag{2}$$

The *Adams-Bashforth* formula of order k, ABk, is $y_{n+1} = P_n(t_{n+1})$. The example (1) is AB3. A very convenient aspect of this family of formulas is that the polynomial $P_n(t)$ is a natural continuous extension. The *Adams-Moulton* formulas are constructed in the same way except that $Q(x)$ also interpolates the unknown value $f(t_{n+1}, y_{n+1})$. This results in implicitly defined formulas such as AM3

$$y_{n+1} = y_n + h\left[\frac{5}{12} f(t_{n+1}, y_{n+1}) + \frac{8}{12} f_n - \frac{1}{12} f_{n-1}\right].$$

The new value y_{n+1} of an implicit Adams-Moulton formula is computed by iteration. In practice, this costs a little less than twice as many function evaluations as an explicit Adams-Bashforth formula. However, the Adams-Moulton formulas are more accurate and more stable, so this is a bargain. The point here, however, is that a natural continuous extension (2) is available for these formulas too.

Another important family of formulas is based on interpolation of previously computed values. The *backward differentiation formulas, BDFs*, are defined by a polynomial $P_n(t)$ that interpolates solution values $y_{n+1}, y_n, \ldots$ and satisfies the differential equation at t_{n+1}, i.e., $P_n'(t_{n+1}) = f(t_{n+1}, P_n(t_{n+1}))$. For instance, BDF3 is

$$hf(t_{n+1}, y_{n+1}) = \frac{11}{6} y_{n+1} - 3y_n + \frac{3}{2} y_{n-1} - \frac{1}{3} y_{n-2}.$$

These formulas are implicit, and evaluating them efficiently is the principal challenge when solving stiff IVPs. Here, however, the point is that these methods are defined in terms of polynomials $P_n(t)$ which are natural continuous extensions.

The formulas exhibited are linear combinations of previously computed values and slopes and in the case of implicit formulas, the value and slope at the next step. They are representative of linear multistep methods, LMMs [9]. By using more data, it is possible to obtain formulas of higher order, but they have serious defects. The Adams methods and closely related methods called *predictor-corrector methods* are very popular for the solution of non-stiff IVPs, and the BDFs are very popular for stiff IVPs. All these methods have natural continuous extensions, which contributes to their popularity. And, from the derivation outlined, it is clear that the methods are defined for mesh points that are not equally spaced. Some popular programs work with constant step size until a change appears worth the cost. This is standard for BDFs, including the `ode15s` program of Shampine and Reichelt [13]. Other programs vary the step size, perhaps at every step. This is less common, but is the case for the Adams-Bashforth-Moulton predictor-corrector method of `ode113` [13]. A continuous extension for other LMMs can be constructed by interpolation to all the values and slopes used by the formula (Hermite interpolation). With some care in the selection of step size, this is a satisfactory continuous extension. Still, only a very few other LMMs are seen in practice. One, the midpoint rule, underlies a popular approach to solving IVPs discussed in the section "Extrapolation Methods."

Runge-Kutta Methods

Using previously computed values causes some difficulties for LMMs. For instance, where do these values come from at the start of the integration? *Runge-Kutta, RK, methods* use only information gathered in the current step, so are called *one-step methods*.

An explicit RK formula of three function evaluations, or *stages*, has the form

$$\begin{aligned} y_{n,1} &= y_n, \\ y_{n,2} &= y_n + h\beta_{2,1} f_{n,1}, \\ y_{n,3} &= y_n + h\left[\beta_{3,1} f_{n,1} + \beta_{3,2} f_{n,2}\right], \\ y_{n+1} &= y_n + h\left[\gamma_1 f_{n,1} + \gamma_2 f_{n,2} + \gamma_3 f_{n,3}\right]. \end{aligned}$$

Here $t_{n,j} = t_n + \alpha_j h$ and $f_{n,j} = f(t_{n,j}, y_{n,j})$. The coefficients $\alpha_j, \beta_{j,k}$, and γ_j are chosen primarily to make y_{n+1} approximate $y(t_{n+1})$ to high order. It is easy to find coefficients that make a LMM as high an order as possible because they appear in a linear way. This is very much more complicated and difficult with RK methods because the coefficients appear in a nonlinear way. The higher the order, the more algebraic equations, the *equations of condition*, and the number increases rapidly with the order. It is actually easy to find formulas of any given order – the trick is to find formulas of few stages. It is known that it takes at least k stages to get a formula of order k. In this case, it is possible to get order 3 with just three stages. Typically RK formulas involve families of parameters, and that is the case for this example.

Explicit RK methods are much more expensive in terms of function evaluations than an explicit Adams method, but they are competitive because they are more accurate. However, for this argument to be valid, a program must be allowed to use the largest step sizes that provide the specified accuracy. As a consequence, it is especially inefficient with RK methods to obtain output at specific points by reducing the step size so as to produce a result at those points. Event location is scarcely practical for RK methods without a continuous extension. Unfortunately, it is much harder to construct continuous extensions for RK methods than for LMMs.

An obvious approach to constructing a continuous extension is to use Hermite polynomial interpolation to $y_n, y_{n-1}, \ldots$ and $f_n, f_{n-1}, \ldots$, much as with LMMs. Gladwell [6] discusses the difficulties that arise when interpolating over just two steps. An important advantage of RK methods is that they do not require a starting procedure like the methods of section "Linear Multistep Methods," but this approach to continuous extension *does* require starting values. Further, convergence of the approximation requires control of the rate of increase of step size. This approach can be used at low orders, but a more fundamental difficulty was recognized as higher-order formulas came into use. In the case of explicit Adams methods, the step size is chosen so that an interpolating polynomial provides an accurate approximation throughout the step. Runge-Kutta methods of even moderate order use much larger step sizes that are chosen independent of any polynomial interpolating at previous steps. In practice, it was found that the interpolating polynomial does not achieve anything like the accuracy of the approximations at mesh points.

The resolution of an important difference between RK methods and LMMs is crucial to the construction of satisfactory continuous extensions. This difference is in the estimation of the error of a step. LMMs can use previously computed values for this purpose. There are several approaches taken to error estimates for RK methods, but they are equivalent to taking each step with two formulas of different orders and estimating the error of the lower-order formula by comparison. RK methods involve a good many stages per step, so to make this practical, the two formulas are constructed so that they share many function evaluations. Generally this is done by starting with a family of formulas and looking for a good formula that uses the same stages and is of one order lower. Fehlberg [5] was the first to introduce these embedded formulas and produce useful pairs. For example, it takes at least six stages to obtain an explicit RK formula of order 5. He found a pair of orders 4 and 5 that requires only the minimum of six stages to evaluate both formulas. Later he developed pairs of higher order [4], including a very efficient (7, 8) pair of 13 stages.

Another matter requires discussion at this point. If each step is taken with two formulas, it is only natural to advance the integration with the higher-order formula provided, of course, that other properties like stability are acceptable. After all, the reliability of the error estimate depends on the higher-order formula being more accurate. In this approach, called *local extrapolation*, the step size is chosen to make the lower-order result pass an error test, but a value believed to be more accurate is used to advance the integration. All the popular programs based on explicit RK methods do local extrapolation. There is a related question about the order of a continuous extension. If the formula used to advance the integration has a local error that is $O(h^{p+1})$, the true, or global, error $y(t_n) - y_n$ is

$O(h^p)$, which is to say that the formula is of order p. Roughly speaking, for a stable problem and formula, errors at each step that are $O(h^{p+1})$ accumulate after $O(1/h)$ steps to yield a uniform error that is $O(h^p)$. This leads us to the question as to the appropriate order of a continuous extension. It would be natural to ask that it has the order of the formula used to advance the integration, but it is not used to propagate the solution, so it can be one lower order and still achieve the global order of accuracy. Because it can be expensive to obtain a continuous extension at high orders, this is an important practical matter.

Horn [10] was the first to present a modern approach to continuous extensions for RK methods. In her approach, a family of formulas is created, one for each point in the span of a step. Each member of the family is a linear combination of the stages used in the basic formula plus other stages as necessary. By virtue of reusing stages, it is possible to approximate the solution anywhere in the span of the step with a small number of extra stages. In more detail, suppose that a total of s stages are formed in evaluating the pair of formulas. For some $0 < \theta < 1$, we approximate the solution at $t_{n+\theta} = t_n + \theta h$ with an explicit RK formula that uses these stages:

$$y_{n+\theta} = y_n + \theta h \left[\gamma_1(\theta) f_{n,1} + \gamma_2(\theta) f_{n,2} + \ldots \gamma_s(\theta) f_{n,s}\right].$$

This is a conventional explicit RK formula of s stages with specified coefficients $\alpha_j, \beta_{j,k}$ for approximating $y(t_n + \theta h)$. We look for coefficients $\gamma_j(\theta)$ which provide an accurate approximation $y_{n+\theta}$. This is comparatively easy because these coefficients appear in a linear way. Although we have described this as finding a family of RK formulas with parameter θ, the coefficients $\gamma_j(\theta)$ turn out to be polynomials in θ, so we have a continuous extension $P_n(\theta)$. It can happen that there is enough information available to obtain approximations that have an order uniform in θ that corresponds to the global order of the method. For instance, the (4, 5) pair due to Dormand and Prince [2] that is implemented in the `ode45` program of MATLAB is used with a continuous extension that is of order 4. We digress to discuss some practical aspects of continuous extensions of RK formulas with this pair as example.

Solutions of IVPs are customarily studied in graphical form in MATLAB, so the output of the solvers is tailored to this. For nearly all the solvers, which implement a wide range of methods, the default output is the set $\{t_n, y_n\}$ chosen by the solver to obtain accurate results efficiently. Generally this provides a smooth graph, but there is an option that computes additional results at a fixed number of equally spaced points in each step using a continuous extension. The (4, 5) pair implemented in `ode45` must take relatively large steps if it is to compete with Adams methods, and correspondingly, a solution component can change significantly in the span of a step. For this reason, results at mesh points alone often do not provide a smooth graph. The default output of this program is not just results at mesh points but results at four equally spaced points in the span of each step. This usually provides a smooth graph. In this context, a continuous extension is formed and evaluated at every step. The pair does not involve many stages, so any additional function evaluations would be a significant expense. This is why a "free" continuous extension of order 4 was chosen for implementation.

Some of the continuous extensions can be derived in a more direct way by interpolation [12] that we use to raise another matter. The $y_{n,j}$ approximate $y(t_{n,j})$, but these approximations are generally of low order. Some information of high order of accuracy is to hand. After forming the result y_{n+1} that will be used to advance the integration, we can form $f(t_{n+1}, y_{n+1})$ for use in a continuous extension. Certainly we would prefer continuous extensions that are not only continuous but also have a continuous derivative from one step to the next. To construct such an extension, we must have f_{n+1}. Fortunately, the first stage of an explicit RK formula is always $f_n = f(t_n, y_n)$, so the value f_{n+1} is "free" in this step because it will be used in the next step. We can then use the cubic Hermite interpolant to value and slope at both ends of the step as continuous extension. Interpolation theory can be used to show that it is an excellent continuous extension for any formula of order no higher than 3. It is used in the MATLAB program `ode23` [13]. Some of the higher-order formulas that have been implemented have one or more intermediate values $y_{n,j}$ that are sufficiently accurate that Hermite interpolation at these values, and the two ends of the step provides satisfactory continuous extensions.

If the stages that are readily available do not lead to a continuous extension that has a sufficiently high order uniformly in $0 \leq \theta \leq 1$, we must somehow obtain additional information that will allow us to achieve our goal. A tactic [3] that has proved useful is to observe

that in addition to the ends of the step, the extension $P_{n,s}(\theta)$ based on s stages may be of higher order at one or more points in $(0, 1)$. If θ^* is such a point, we define $t_{n,s+1} = t_n + \theta^* h$ and $y_{n,s+1} = y_{n+\theta^*}$ and evaluate $f_{n,s+1} = f(t_{n,s+1}, y_{n,s+1})$. If there is more than one such point, we do this for each of the points. We now try to find a continuous extension that uses these new stages in addition to the ones previously formed. If this new continuous extension has a uniform order that is acceptable, we are done and otherwise we repeat. This tactic has resulted in continuous extensions for some popular formulas of relatively high order. After Fehlberg showed the effectiveness of a (7, 8) pair of 13 stages, several authors produced pairs that are better in some respects and more to the point have continuous extensions. Current information about quality RK pairs is found at Verner [15]. Included there are (7, 8) pairs with a continuous extension of order 7 that requires three additional stages and order 8 that requires four.

Implicit Runge-Kutta, IRK, formulas are exemplified by the two-stage formula

$$\begin{aligned} y_{n,1} &= y_n + h\left[\beta_{1,1} f_{n,1} + \beta_{1,2} f_{n,2}\right], \\ y_{n,2} &= y_n + h\left[\beta_{2,1} f_{n,1} + \beta_{2,2} f_{n,2}\right], \\ y_{n+1} &= y_n + h\left[\gamma_1 f_{n,1} + \gamma_2 f_{n,2}\right]. \end{aligned}$$

This is a pair of simultaneous algebraic equations for $y_{n,1}$ and $y_{n,2}$, and as a consequence, it is much more trouble to evaluate an IRK than an explicit RK formula. On the other hand, they can be much more accurate. Indeed, if $t_{n,1}$ and $t_{n,2}$ are the nodes of the two-point Gauss-Legendre quadrature formula shifted to the interval $[t_n, t_{n+1}]$, the other coefficients can be chosen to achieve order 4. For non-stiff IVPs, this high order is not worth the cost. However, IRKs can also be very much more stable. Indeed, the two-stage Gaussian formula is A-stable. This makes them attractive for stiff problems despite the high costs of evaluating the formulas for such problems. IRKs are also commonly used to solve boundary value problems for ODEs. IVPs specify a solution of a set of ODEs by the value $y(t_0)$ at the initial point of the interval $t_0 \le t \le t_F$. Two-point *boundary value problems, BVPs*, specify a solution by means of values of components of the solution at the two ends of the interval. More specifically, the vector solution $y(t)$ is to satisfy a set of equations, $g(y(t_0), y(t_F)) = 0$. In this context, the formula must be evaluated on all subintervals $[t_n, t_{n+1}]$ simultaneously. This is typically a large system of nonlinear equations that is solved by an iterative procedure. If an approximation to $y(t)$ is not satisfactory, the mesh is refined and a larger system of algebraic equations is solved. A continuous extension is fundamental to this computation because it is used to generate starting guesses for the iterative procedure.

The IRKs commonly implemented are based on Gaussian quadrature methods or equivalently *collocation*. There is a sense of direction with IVPs, so the formulas for stiff IVPs in wide use are based on Radau formulas. The lowest-order case is the implicit backward Euler method $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$, a formula that happens to be AM1 and BDF1. There is no preferred direction when solving BVPs with implicit RK methods, so the symmetric Gauss-Legendre or Gauss-Lobatto formulas are used. The nodes of the former do not include an endpoint of the step, and the nodes of the latter include both. As mentioned above, the two-point Gauss-Legendre formula is of order 4. It can be derived by collocation rather like the BDFs. This particular formula is equivalent to collocation with a quadratic polynomial $P_n(t)$ that interpolates $P_n(t_n) = y_n$ and also $P(t_{n,j}) = y_{n,j}$ for $j = 1, 2$. The $y_{n,j}$ are determined by the collocation conditions $P'_n(t_{n,j}) = f(t_{n,j}, P(t_{n,j}))$ for $j = 1, 2$. Although the formula is of order 4 at mesh points, this quadratic approximation has a uniform order of 2. This is typical. Popular codes like COLSYS [1] use Gauss-Legendre formulas of quite high order for which the uniform order of approximation by the collocation polynomial is roughly half the order of approximation at mesh points. This is not all that one might hope for, but a convenient continuous extension is very important and formulas of a wide range of orders are available. The three-point Gauss-Lobatto formula collocates at both endpoints of the step and the midpoint. The underlying cubic polynomial is uniformly of order 4, which is adequate for solving BVPs in MATLAB. The collocation conditions imply that the approximation is $C^1[t_0, t_F]$, which is useful in a computing environment where results are often studied graphically.

Extrapolation Methods

Extrapolation methods are built upon relatively simple methods of order 1 or 2 such as the explicit/forward Euler method and the implicit midpoint rule.

The construction of extrapolation methods relies on the theoretical existence of an asymptotic expansion of the global error of the low-order underlying method in terms of a constant step size h. For example, let us consider the explicit/forward Euler method

$$y_{k+1} = y_k + hf(t_k, y_k).$$

We denote $y_h(t_k + nh) = y_{k+n}$ for $n = 0, 1, 2, \ldots$ With initial condition $y(t_k) = y_k$, it can be shown that for sufficiently smooth $f(t, y)$, the global error at $t = t_k + nh$ of the explicit Euler method has an asymptotic expansion of the form

$$y_h(t) - y(t) = e_1(t)h + e_2(t)h^2 + \ldots + e_N(t)h^N + E_N(t, h)h^{N+1},$$

where $e_1(t), \ldots, e_N(t)$ are smooth functions and $E_N(t, h)$ is bounded for $|h|$ sufficiently small. For a symmetric method such as the implicit midpoint rule, all the odd terms $e_{2k+1}(t)$ vanish. Given a finite sequence of increasing natural numbers n_i for $i = 1, \ldots, I$ such as $n_i = i$, for a given macro step size H, we define the micro step sizes $h_i = H/n_i$. By independent applications of n_i steps of the explicit Euler method with constant step size h_i, we obtain a finite sequence $Y_{i1} = y_{h_i}(t_k + H)$ of approximations to the solution $y(t_k + H)$ of the ODE passing through $y(t_k) = y_k$. Defining the table of values

$$Y_{i,j+1} = Y_{ij} + \frac{Y_{ij} - Y_{i-1,j}}{(n_i/n_{i-j}) - 1} \quad \text{for } i = 2, \ldots, I,\ j = 1, \ldots, i-1, \qquad (3)$$

the extrapolated values Y_{ij} are of order j, i.e., $Y_{ij} - y(t_k + H) = O\left(H^{j+1}\right)$. For symmetric methods, we replace the term n_i/n_{i-j} in (3) by $(n_i/n_{i-j})^2$, and we obtain $Y_{ij} - y(t_k + H) = O\left(H^{2j+1}\right)$. If there is no stiffness, an efficient symmetric extrapolation method is given by the Gragg-Bulirsch-Stoer (GBS) algorithm where $y_h(t_k + nh)$ for $n \geq 2$ with n even is obtained starting from $z_0 = y_k$ as follows:

$$z_1 = z_0 + hf(t_k, z_0), \quad z_{l+1} = z_{l-1} + 2hf(t_k + lh, z_l) \quad \text{for } l = 1, \ldots, n,$$
$$y_h(t_k + nh) = \frac{1}{4}(z_{n-1} + 2z_n + z_{n+1}).$$

Due to their possible high-order, extrapolation methods may take large step sizes H. Hence, the use of a sufficiently high order continuous extension is really required if an accurate approximation at intermediate points is needed. A continuous extension can be obtained by building a polynomial approximation to the solution. First finite-difference approximations $D_{1i}^{(m)}(t)$ to the derivatives $y^{(m)}(t)$ at the left endpoint $t = t_k$, at the midpoint $t = t_k + H/2$, or/and at the right endpoint $t = t_k + H$ are built for each index i when possible based on the intermediate values of $f(t, y)$ or y. In the presence of stiffness, it is not recommended to use the intermediate values based on $f(t, y)$ since f may amplify errors catastrophically, and approximations to the derivatives should be based only on the intermediate values of y in this situation. The values $D_{1i}^{(m)}(t)$ are extrapolated to obtain higher-order approximations. We denote the most extrapolated value by $D^{(m)}(t)$. A polynomial $P(\theta)$ approximating $f(t_k + \theta H)$ is then defined through Hermite interpolation conditions. For example, for the GBS algorithm, we consider a sequence of increasing even natural numbers n_i satisfying $n_{i+1} \equiv n_i \mod 4$. We define a polynomial $P_d(\theta)$ of degree $d + 4$ with $-1 \leq d \leq 2I$ satisfying the Hermite interpolation conditions

$$P_d(0) = y_k, \quad P_d(1) = Y_{II}, \quad P_d'(0) = Hf(t_k, y_k),$$
$$P_d'(1) = Hf(t_k + H, Y_{II}),$$
$$P_d^{(m)}(1/2) = H^m D^{(m)}(t_k + H/2) \text{ for } m = 0, \ldots, d.$$

For $n_1 = 4$ and $d \geq 2I - 4$, it can be shown that $P_d(\theta)$ is an approximation of order $2I$ in H to $y(t_k + \theta H)$, i.e., $P_d(\theta) - y(t_k + \theta H) = O(H^{2I+1})$ for $\theta \in [0, 1]$.

If one wants to have a continuous extension with a certain required accuracy, one also needs to control its error and not just the error at the endpoint. This can be done by using an upper bound on the norm of the difference between the continuous extension and another continuous extension of lower order. For example, for the GBS algorithm for $d \geq 0$, one can consider the difference

$$P_d(\theta) - P_{d-1}(\theta) = \theta^2(1 - \theta)^2(\theta - 1/2)^d c_{d+4},$$

where c_{d+4} is the coefficient of θ^{d+4} in $P_d(\theta)$. The function $|\theta^2(1 - \theta)^2(\theta - 1/2)^d|$ is maximum on $[0, 1]$

at $\theta_d = \frac{1}{2}(1 \pm \sqrt{d/(d+4)})$, and we obtain the error estimate

$$\max_{\theta\in[0,1]} \|P_d(\theta) - P_{d-1}(\theta)\| \le |\theta_d^2(1-\theta_d)^2 (\theta_d - 1/2)^d| \cdot \|c_{d+4}\|,$$

which can be used in a step size controller.

For more information on continuous extensions for extrapolation methods, we refer the reader to Hairer and Ostermann [7] for the extrapolated Euler method and the linearly implicit Euler method; to Hairer and Ostermann [7], Hairer et al. [8], and Shampine et al. [14] for the GBS algorithm; and to Jay [11] for the semi-implicit midpoint rule.

References

1. Ascher, U.M., Christiansen, J., Russell, R.D.: Collocation software for boundary value ODEs. ACM Trans. Math. Softw. **7**, 209–222 (1981)
2. Dormand, J.R., Prince, P.J.: A family of embedded Runge-Kutta formulae. J. Comput. Appl. Math. **6**, 19–26 (1980)
3. Enright, W.H., Jackson, K.R., Nørsett, S.P., Thomson, P.G.: Interpolants for Runge-Kutta formulas. ACM Trans. Math. Softw. **12**, 193–218 (1986)
4. Fehlberg, E.: Classical fifth-, sixth-, seventh-, and eighth order Runge-Kutta formulas with step size control. Technical report, 287, NASA (1968)
5. Fehlberg, E.: Klassische Runge-Kutta-Formeln vierter und niedrigerer Ordnung mit Schrittweiten-Kontrolle und ihre Anwendung auf Wärmeleitungsprobleme. Computing **6**, 61–71 (1970)
6. Gladwell, I.: Initial value routines in the NAG library. ACM Trans. Math. Softw. **5**, 386–400 (1979)
7. Hairer, E., Ostermann, A.: Dense output for extrapolation methods. Numer. Math. **58**, 419–439 (1990)
8. Hairer, E., Nørsett, S.P., Wanner, G.: Solving Ordinary Differential Equations I. Nonstiff Problems. Springer Series in Computational Mathematics, vol. 18, 2nd edn. Springer, Berlin (1993)
9. Henrici, P.: Discrete Variable Methods in Ordinary Differential Equations. Wiley, New York (1962)
10. Horn, M.K.: Fourth and fifth-order scaled Runge-Kutta algorithms for treating dense output. SIAM J. Numer. Anal. **20**, 558–568 (1983)
11. Jay, L.O.: Dense output for extrapolation based on the semi-implicit midpoint rule. Z. Angew. Math. Mech. **73**, 325–329 (1993)
12. Shampine, L.F.: Interpolation for Runge-Kutta methods. SIAM J. Numer. Anal. **22**, 1014–1027 (1985)
13. Shampine, L.F., Reichelt, M.W.: The MATLAB ODE suite. SIAM J. Sci. Comput. **18**, 1–22 (1997)
14. Shampine, L.F., Baca, L.S., Bauer, H.J.: Output in extrapolation codes. Comput. Math. Appl. **9**, 245–255 (1983)
15. Verner, J.: Jim Verner's refuge for Runge-Kutta pairs. http://people.math.sfu.ca/~jverner/ (2011)

Density Functional Theory

Rafael D. Benguria
Departamento de Física, Pontificia Universidad Católica de Chile, Santiago de Chile, Chile

Synonyms

Exchange corrections; Generalized gradient corrections; Kohn–Sham equations; Local density approximation; Statistical model of atoms; Thomas–Fermi

Definition

Density functional theory (DFT for short) is a powerful, widely used method for computing approximations of ground state electronic energies and densities in chemistry, material science, and biology. The purpose of DFT is to express the ground state energy (as well as many other quantities of physical and chemical interest) of a multiparticle system as a functional of the single-particle density ρ_ψ.

Overview

Since the advent of quantum mechanics [20], the impossibility of solving exactly problems involving many particles has been clear. These problems are of interest in such areas as atomic and molecular physics, condensed matter physics, and nuclear physics. It was, therefore, necessary from the early beginnings to introduce approximative methods such as the Thomas–Fermi model [4, 21], (see J. P. Solovej ▸ Thomas–Fermi Type Theories (and Their Relation to Exact Models) in this encyclopedia) and the Hartree–Fock approximation [5, 6] (see I. Catto ▸ Hartree–Fock Type Methods in this encyclopedia), to compute quantities of physical interest in these areas. In quantum mechanics of many particle systems,