Ordinary Differential Equations I

A. Godunov

1. Overview
2. Finite difference approximation
3. First-order ODE: Single-point methods
   a) Euler and b) Runge-Kutta methods
4. Additional information

1

Part 1:

Overview

2

Motivation

Most of equations in physics are formulated in terms of ordinary differential equations (ODE) or partial differential equations (PDE).

Examples:

Newton's second law

$$\frac{d\vec{p}}{dt} = \vec{F}$$

Schrodinger equation for a particle in a potential

$$i\hbar\frac{\partial}{\partial x}\psi(\vec{r},t) = \left(-\frac{\hbar^2}{2m}\Delta + V(\vec{r})\right)\psi(\vec{r},t)$$

Most real physics processes involve more than one independent variable, and the corresponding equations are partial differential equations. In many cases, however, physics can be represented by ordinary differential equations, or PDEs can be reduced to ODEs.

We will concentrate on solutions of ordinary differential equations.

3

3

ODEs and PDEs

Most real physics processes involve more than one independent variable, and the corresponding equations are partial differential equations.

In many cases, however, physics can be represented by ordinary differential equations, or PDEs can be reduced to ODEs.

We will concentrate on solutions of ordinary differential equations.

4

4

Major categories of ODEs

1. Initial value problems
   Conditions for the unknown function are specified at the same point.
   example: $x(t_0) = x_0,\ x'(t_0) = v_0$
2. Boundary value problems
   Conditions for the unknown function are specified at different boundaries.
   example: $y(a) = y_a,\ \ y(b) = y_b$
3. Eigenvalue problems
   A special type of boundary value problems, when solutions exists only for specific values of parameters.

5

5

Classification of physical problems

Physical problems fall into one of the following three general classifications:

1. Propagation problems are initial-value problems in open domains in which the known information (initial values) are marched forward in time or space from the initial state. The order of ODE may be one or greater. The number of initial values must be equal to the order of the differential equation.
2. Equilibrium problems are boundary-value problems in closed domains in which the known information (boundary values) are specified at two different values of the independent variable, the end points (boundaries) of the solution domain. The order of the governing differential equation must be at least 2, and may be greater. The number of boundary values must be equal to the order of the differential equation. Equilibrium problems are steady state problems in closed domains.
3. Eigenproblems are a special type of problem in which the solution exists only for special values (i.e., eigenvalues) of a parameter of the problem. The eigenvalues are to be determined in addition to the corresponding solutions of the system.

6

6

Part 2:

Finite difference approximation

7

March, march, march …

Initial-value ODEs are solved numerically by marching methods.

We will concentrate on finite difference methods.

The objective of a **finite difference method** for solving an ODE is to transform a calculus problem into an algebra problem by

1. Discretizing the continuous physical domain into a discrete finite difference grid
2. Approximating the exact derivatives in the ODE by algebraic finite difference approximations (FDAs)
3. Substituting the FDA into ODE to obtain an algebraic finite difference equation (FDE)
4. Solving the resulting algebraic FDE

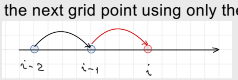
8

8

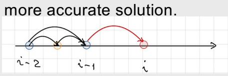
Three groups of the Finite Difference Methods

Three groups of finite difference methods for solving initial-value ODEs

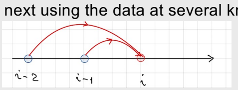
1. Single point methods advance the solution from one grid point to the next grid point using only the data at a single grid point.



2. Extrapolation methods evaluate the solution at a grid point for several values of grid size and extrapolate those results to get a more accurate solution.



3. Multipoint methods advance the solution from one grid point to the next using the data at several known points



9

9

Finite difference approximation

In the development of finite difference approximations of differential equations, a distinction must be made between the exact solution of the differential equation and the solution of the finite difference equation which is an approximation of the exact differential equation.

Notations:

$x(t)$ – exact solution

$x(t)$ – approximate solution

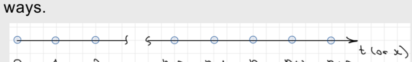
This very precise distinction between the exact solution of a differential equation and the approximate solution of a differential equation is required for studies of consistency, order, stability, and convergence

10

10

Finite difference approximation

Exact derivatives, such as $x'(t)$, can be approximated at a grid point in terms of the values of $x(t)$ at that grid point and adjacent grid points in several ways.



Writing the Taylor series for x_{n+1} using grid point n as the base gives

$$x_{n+1} = x_n + x_n' \Delta t + \frac{1}{2} x_n'' \Delta t^2 + \frac{1}{6} x_n''' \Delta t^3 + \cdots$$

Solving for x_n' gives

$$x_n' = \frac{x_{n+1} - x_n}{\Delta t} - \frac{1}{2} x_n'' \Delta t - \frac{1}{6} x_n''' \Delta t^2 - \cdots$$

If equation is terminated after the first term, it becomes FDA of x_n'

$$x_n' = \frac{x_{n+1} - x_n}{\Delta t} \qquad O(\Delta t)$$

11

11

Finite difference approximation (cont.)

A first-order forward-difference approximation of x_n' at grid point n

$$x_n' = \frac{x_{n+1} - x_n}{\Delta t} \qquad O(\Delta t)$$

A first-order backward-difference approximation of x_{n+1}' at grid point n

$$x_{n+1}' = \frac{x_{n+1} - x_n}{\Delta t} \qquad O(\Delta t)$$

where the $O(\Delta t)$ term is the order of the remainder term which was truncated, which is the order of the approximation of x_n'.

A second-order centered-difference approximation of x_n' at grid point n

$$x_n' = \frac{x_{n+1} - x_{n-1}}{2\Delta t} \qquad O(\Delta t^2)$$

12

12

Finite difference equations

Finite difference solutions of differential equations are obtained by discretizing the continuous solution domain and replacing the exact derivatives in the differential equation by finite difference approximations.

Such approximations are called finite difference equations (FDEs).

Example: consider

$$\frac{dx}{dt} = f(x,t)$$

Using

$$x_n' = \frac{x_{n+1} - x_n}{\Delta t}$$

yields

$$x_{n+1} = x_n + f(x_n, t_n)\Delta t$$

13

13

Finite difference equations and smoothness

Smoothness refers to the continuity of a function and its derivatives.

The finite difference method of solving a differential equation employs Taylor series to develop finite difference approximations (FDAs) of the exact derivatives in the differential equation.

If a problem has discontinuous derivatives of some order at some point in the solution domain, then FDAs based on the Taylor series may misbehave at that point.

For example, consider the vertical flight of a rocket. When the rocket engine is turned off, the thrust drops to zero instantly. This causes a discontinuity in the acceleration of the rocket, which causes a discontinuity in the second derivative of the altitude $y(t)$.

At a discontinuity – use either single point methods or extrapolation methods (not multi-point) because the step size can be chosen to have the discontinuity at a grid point.

14

14

Consistency, order, stability, and convergence

There are several important concepts which must be considered when developing finite difference approximations of initial-value differential equations. They are a) consistency, b) order, c) stability, and d) convergence.

a) A FDE is consistent with an ODE if the difference between them (i.e., the truncation error) vanishes as $\Delta t \to 0$. In other words, the FDE approaches the ODE.

b) The order of a FDE is the rate at which the global error decreases as the grid size approaches zero.

c) A FDE is stable if it produces a bounded solution for a stable ODE and is unstable if it produces an unbounded solution for a stable ODE.

d) A finite difference method is *convergent* if the numerical solution of the FDE (i.e., the numerical values) approaches the exact solution of the ODE as $\Delta t \to 0$.

15

15

Part 3a:

Single-point methods

Euler methods

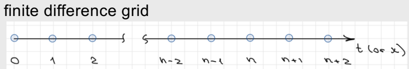
16

The explicit Euler method

Consider the general non-linear first-order ODE

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

with the finite difference grid



Choose point n as the base point and use the first-order forward-difference finite difference approximation for x_n'

$$x_n' = \frac{x_{n+1} - x_n}{\Delta t} = f(x_n, t_n)$$

Explicit Euler finite difference equation

$$x_{n+1} = x_n + f(x_n, t_n)\Delta t \qquad O(\Delta t^2)$$

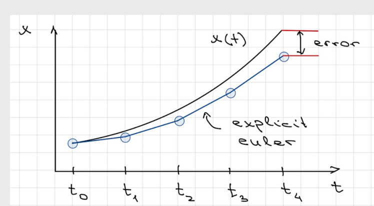
where the $O(\Delta t^2)$ term is included as a reminder of the order of the local truncation error.

17

17

The explicit Euler method

The linear extrapolation with the slope



18

18

The explicit Euler method (summary)

$$x_{n+1} = x_n + f(x_n, t_n)\Delta t$$

1. The method is explicit since, $f(x_n, t_n)$ does not depend on x_{n+1}
2. The method requires only one known point. Hence it is a single point method.
3. A single FDE is required to advance the solution from n to $n+1$. Thus, the method is a single-step method.
4. The method requires only one derivative function evaluation ,i.e., $f(x_n, t_n)$ per step.
5. The error in calculating x_{n+1} for a single step, the local truncation error, is $O(\Delta t^2)$
6. The global (i.e. total) error accumulated after n steps is $O(\Delta t)$, which is the same order as FDA of the exact derivative $x'(t)$.

19

19

Example: C++ explicit Euler method

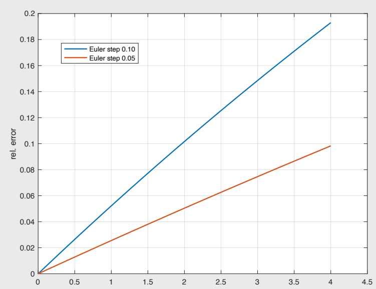
```
/*--------------------------------------------------------------------
 Program to solve 1st order Ordinary Differential Equations
 x'(t) = f(t,x)
 method:    simple Euler method
 input ...
 f(t,x)- function supplied by a user
 ti  - initial value for an independent variable (t)
 xi  - initial value for a function x(t)
 tf  - find solution for this point t
 output ...
 xf  - solution at point tf, i.e. x(tf)
--------------------------------------------------------------------*/
double euler1d(double(*f)(double, double), double ti, double xi, double tf)
{
   double xf;
     xf = xi + f(ti,xi)*(tf-ti);
   return xf;
}

see full code at: https://ww2.odu.edu/~agodunov/book/programs.html
```

20

20

Example: relative error for two time steps

$x'(t) = -x \quad x(t) = e^{-t}, \qquad x(0) = 1;$



21

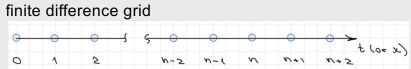
21

The implicit Euler method

Consider (again) the general non-linear first-order ODE

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

with the finite difference grid



Choose point $n+1$ as the base point and use the first-order backward-difference finite difference approximation for x'_n

$$x'_{n+1} = \frac{x_{n+1} - x_n}{\Delta t} = f(x_{n+1}, t_{n+1})$$

Implicit Euler finite difference equation

$$x_{n+1} = x_n + f(x_{n+1}, t_{n+1})\Delta t \qquad O(\Delta t^2)$$

where the $O(\Delta t^2)$ term is included as a reminder of the order of the local truncation error.

22

22

The implicit Euler method (summary)

$$x_{n+1} = x_n + f(x_{n+1}, t_{n+1})\Delta t$$

1. The method implicit since, $f(x_{n+1}, t_{n+1})$ depends on x_{n+1}. If $f(x,t)$ is linear in x, then f_{n+1} is linear in x_{n+1}, and equation above can be solved directly for x_{n+1}. If $f(x,t)$ is nonlinear in x, we deal with a nonlinear FDE, and additional effort is required to solve a non-linear equation for x_{n+1}.
2. The method is a single point method.
3. The FDE requires only one derivative function evaluation per step if $f(x,t)$ is linear in x. If $f(x,t)$ is nonlinear then several evaluations of the derivative function may be required to solve the nonlinear FDE.
4. The single step truncation error is $O(\Delta t^2)$, and the global (i.e. total) error accumulated after n steps is $O(\Delta t)$.

23

23

Comparison of the explicit and implicit Euler methods

The explicit Euler method and the implicit Euler method are both first-order $O(\Delta t)$ methods. The errors in these two methods are comparable (although of opposite sign) for the same step size.

For nonlinear ODEs, the explicit Euler method is straightforward, but the implicit Euler method yields a nonlinear FDE, which is more difficult to solve.

So what is the advantage, if any, of the implicit Euler method?

24

24

Comparison of the explicit and implicit Euler methods

The explicit Euler method and the implicit Euler method are both first-order $O(\Delta t)$ methods. The errors in these two methods are comparable (although of opposite sign) for the same step size.

For nonlinear ODEs, the explicit Euler method is straightforward, but the implicit Euler method yields a nonlinear FDE, which is more difficult to solve.

So what is the advantage, if any, of the implicit Euler method?

Let us consider ODE

$$\frac{dx}{dt} + x = 0, \qquad x(0) = 1, \qquad x(t) = e^{-t}$$

Explicit Euler $x_{n+1} = x_n + f(x_n, t_n)\Delta t$ gives $x_{n+1} = (1 - \Delta t)x_n$

Implicit Euler $x_{n+1} = x_n + f(x_{n+1}, t_{n+1})\Delta t$ gives: $x_{n+1} = \frac{x_n}{1+\Delta t}$

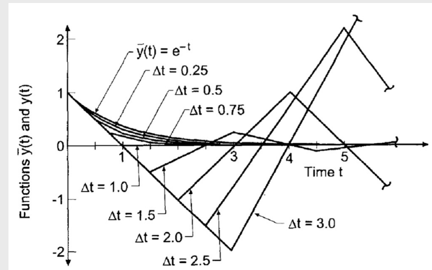
25

25

Stability

Explicit Euler

$$x_{n+1} = (1 - \Delta t)x_n$$



For $\Delta t > 2.0$ the numerical solution oscillates and growth exponentially!

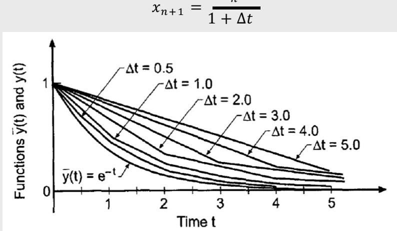
26

26

Stability

Implicit Euler yields

$$x_{n+1} = \frac{x_n}{1 + \Delta t}$$



The error increases as Δt increases, but this is accuracy problem, not a stability problem.

27

27

Stability (summary)

The implicit Euler method is unconditionally stable, whereas the explicit Euler method is conditionally stable.

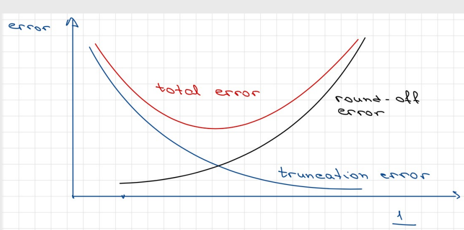
28

28

Truncation error

For both methods the total error is $O(\Delta t)$.

What if we make Δt smaller and smaller?



29

29

Modified Euler: predictor-corrector method.

- First-order explicit Euler method leads to simple FDE. However the method is conditionally stable.
- Implicit first-order order Euler method leads to nonlinear equations, however it is unconditionally stable.
- Predictor-corrector method maintains advantages of both methods (linear FDEs and unconditionally stable) while removing the disadvantages of both methods (conditional stability and non-linear FDEs).

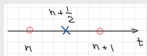
30

30

Modified Euler: predictor-corrector method.

Consider (again) the general non-linear first-order ODE

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



Choosing $n + 1/2$ as a base point

$$x_{n+1} = x_{n+1/2} + x'_{n+1/2}\frac{\Delta t}{2} + \frac{1}{2}x''_{n+1/2}\frac{\Delta t^2}{2^2} + \frac{1}{6}x'''_{n+1/2}\frac{\Delta t^3}{2^3} + \cdots$$

$$x_n = x_{n+1/2} - x'_{n+1/2}\frac{\Delta t}{2} + \frac{1}{2}x''_{n+1/2}\frac{\Delta t^2}{2^2} - \frac{1}{6}x'''_{n+1/2}\frac{\Delta t^3}{2^3} + \cdots$$

The difference

$$x'_{n+1/2} = \frac{x_{n+1} - x_n}{\Delta t} + O(\Delta t^3)$$

Dropping the $O(\Delta t^3)$ terms gives

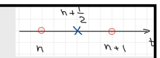
$$x_{n+1} = x_n + f(x_{n+1/2}, t_{n+1/2})\Delta t$$

Equation is the implicit mid-point finite difference equation

31

31

Modified mid-point



$$x_{n+1} = x_n + f(x_{n+1/2}, t_{n+1/2})\Delta t$$

Two unknowns: x_{n+1} and $x_{n+1/2}$

Method 1: modified mid-point

step 1: using Euler method for $x_{n+1/2}$

$$x^{S1}_{n+1/2} = x_n + \frac{1}{2}\Delta t\, f(x_n, t_n)$$

step 2:

$$x^{S2}_{n+1} = x_n + f(x^{S1}_{n+1/2}, t_n + \Delta t/2)\Delta t$$

32

32

Accuracy analysis

Let us consider ODE (again)

$$\frac{dx}{dt} = ax, \qquad x(0) = 1, \qquad x(t) = e^{-at}$$

$$x^{S1}_{n+1/2} = x_n + \frac{1}{2}\Delta t\, f(x_n, t_n) = x_n\left(1 + \frac{1}{2}a\Delta t\right)$$

Substituting into

$$x^{S2}_{n+1} = x_n + f\left(x^{S1}_{n+1/2}, t_n + \frac{\Delta t}{2}\right)\Delta t = x_n + ax_n\left(1 + \frac{1}{2}a\Delta t\right)\Delta t$$

gives

$$x^{S2}_{n+1} = x_n\left(1 + a\Delta t + \frac{1}{2}a^2\Delta t^2\right)$$

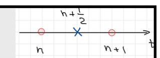
Where $\frac{1}{2}a^2\Delta t^2$ is a correction to explicit Euler

The local truncation error is $O(\Delta t^3)$

33

33

Modified Euler



$$x_{n+1} = x_n + f(x_{n+1/2}, t_{n+1/2})\Delta t$$

Two unknowns: x_{n+1} and $x_{n+1/2}$

Writing Taylor series for. f_{n+1} and f_n using $f_{n+1/2}$ as the base point and adding together yields

$$f_{n+1/2} = \frac{1}{2}(f_{n+1} + f_n) + O(\Delta t^2)$$

Then

$$x_{n+1} = x_n + \frac{1}{2}(f_{n+1} + f_n)\Delta t + O(\Delta t^3)$$

But, we still don't know f_{n+1}.

For linear ODEs equation it can be solved directly, but for non-linear ODEs must be solved iteratively.

However, there is a way …

34

34

Modified Euler

$$x_{n+1} = x_n + \frac{1}{2}(f_{n+1} + f_n)\Delta t + O(\Delta t^3)$$

Step 1: predict x_{n+1} using first-order explicit Euler

$$x^P_{n+1} = x_n + f(x_n, t_n)\Delta t$$

Step 2: Correct x_{n+1}

$$x^C_{n+1} = x_n + \frac{1}{2}\left(f(x^P_{n+1}, t_{n+1}) + f(x_n, t_n)\right)\Delta t$$

This two-step method is known as the modified Euler predictor-corrector method.

35

35

Accuracy analysis

Let us consider ODE (again)

$$\frac{dx}{dt} = ax, \qquad x(0) = 1, \qquad x(t) = e^{-at}$$

$$x^P_{n+1} = x_n + \frac{1}{2}ax_n\Delta t = x_n(1 + a\Delta t)$$

Substituting into

$$x^C_{n+1} = x_n + \frac{1}{2}\left(f(x^P_{n+1}, t_{n+1}) + f(x_n, t_n)\right)\Delta t$$

gives

$$x^C_{n+1} = x_n + \frac{1}{2}\left(ax_n + ax_n(1 + a\Delta t)\right)\Delta t = x_n\left(1 + a\Delta t + \frac{1}{2}a^2\Delta t^2\right)$$

The local truncation error is $O(\Delta t^3)$

And we have the same answer as the modified mid-point method!

36

36

Summary for the modifier Euler method

$$x_{n+1}^{P} = x_n + f(x_n, t_n)\Delta t$$

$$x_{n+1}^{C} = x_n + \frac{1}{2}\left(f(x_{n+1}^{P}, t_{n+1}) + f(x_n, t_n)\right)\Delta t$$

1. The FDE is an explicit predictor-corrector set
2. The method is a single-point, two-step, predictor-corrector method
3. The FDE's global error is $O(\Delta t^2)$
4. The FDE is consistent and conditionally stable, and thus, convergent.

37

37

Example: C++ predictor corrector

```
/*-----------------------------------------------------------------------
 Program to solve 1st order Ordinary Differential Equations
 x'(t) = f(t,x)
 method:    modified Euler method (predictor-corrector)
 input ...
 f(t,x)- function supplied by a user
 ti  - initial value for an independent variable (t)
 xi  - initial value for a function x(t)
 tf  - find solution for this point t
 output ...
 xf  - solution at point tf, i.e. x(tf)
-----------------------------------------------------------------------*/
double euler1m(double(*f)(double, double), double ti, double xi, double tf)
{
   double xf;
     xf = xi + f(ti,xi)*(tf-ti);
     xf = xi + (f(ti,xi)+f(tf,xf))*0.5*(tf-ti);
   return xf;
}

see full code at: https://ww2.odu.edu/~agodunov/book/programs.html
```

38

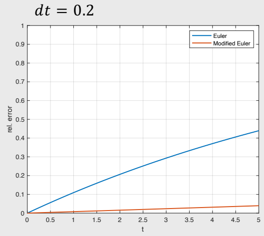
38

Example: compare explicit Euler and predictor-corrector

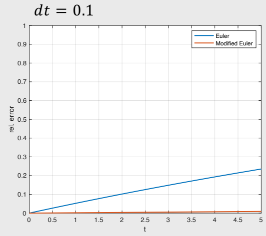
$x'(t) = -x \quad x(t) = e^{-t}, \qquad x(0) = 1;$

for explicit Euler and predictor-corrector

$dt = 0.2$



$dt = 0.1$



39

39

Part 3b:

Single-point methods

Runge-Kutta methods

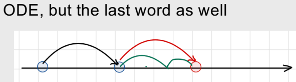
40

Runge-Kutta methods

Runge-Kutta methods are a family of single point multi-step methods.

Runge-Kutta methods propagate a solution over an interval by combining information from several Euler-style steps, and then using the information obtained to match a Taylor series expansion up to some order.

For many scientific users, fourth-order Runge-Kutta is not just the first word on solving ODE, but the last word as well



41

41

Basic idea: $x'(t) = f(x, t)$

Assume that $x_{n+1} - x_n$ can be written as a weighted sum of several Δx_i, where each Δx_i is evaluated as Δt multiplied by the derivative function $f(x, t)$, evaluated at some point in the range $t_n \le t \le t_{n+1}$ and C_i are the weighting factors. Thus,

$$x_{n+1} - x_n = C_1\Delta x_1 + C_2\Delta x_2 + C_3\Delta x_3 + \cdots$$

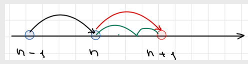
42

42

Second-order Runge-Kutta: $x'(t) = f(x,t)$

Assume that

$x_{n+1} = x_n + C_1 \Delta x_1 + C_2 \Delta x_2$



let $\Delta x_1 = f(x_n, t_n)\Delta t$ explicit Euler

let $\Delta x_2 = f(x_n + \beta \Delta x_1, t_n + \alpha \Delta t)\Delta t$, is evaluated between $t_n \le t \le t_{n+1}$ and α and β are to be determined.

Thus, we have four unknowns: C_1, C_2, α, β.

Let $\Delta t = h$ (easier notation)

$$x_{n+1} = x_n + C_1 h f_n + C_2 h f(x_n + \beta \Delta x_1, t_n + \alpha h)$$

Expressing $f(x,t)$ in a Taylor series at grid point n gives

$$f(x,t) = f_n + \left(\frac{d}{dt} f_n\right)\Delta t + \left(\frac{d}{dx} f_n\right)\Delta x + \cdots$$

Evaluating $f(x,t)$ at $t = t_n + \alpha h$ (i.e. $\Delta t = \alpha h$)

and $x = x_n + \beta \Delta x_1 = x_n + \beta f_n \Delta t$ (i.e. $\Delta x = \beta f_n \Delta t$) gives

43

43

Second-order Runge-Kutta (cont.)

from the slide before: $\Delta t = \alpha h$ and $\Delta x = \beta f_n h$

$$f(x_n + \beta f_n \Delta t, t_n + \alpha h) = f_n + \alpha h f_t'\Big|_n + (\beta h f_n) f_x'\Big|_n + O(h^2)$$

then

$x_{n+1} = x_n + C_1 h f_n + C_2 h f(x_n + \beta \Delta x_1, t_n + \alpha h) =$

$x_{n+1} = x_n + C_1 h f_n + C_2 h \left(f_n + \alpha h f_t'\Big|_n + (\beta h f_n) f_x'\Big|_n\right) + O(h^3)$

Now we match it to the Taylor series for $x(t)$ through second-term

$x_{n+1} = x_n + x_n' h + \frac{1}{2} x_n'' h^2 + \cdots$

$x_n' = f(x_n, t_n) = f_n$

$x_n'' = f_n' = \frac{df}{dt}\Big|_n = f_t'\Big|_n h + f_x'\Big|_n \frac{dx}{dt}\Big|_n = f_t'\Big|_n h + f_x'\Big|_n f_n$

$x_{n+1} = x_n + f_n h + \frac{1}{2} h^2 (f_t' + f_x' f_n)$ where the derivatives are evaluated at n.

44

Second-order Runge-Kutta (cont.)

Equating

$x_{n+1} = x_n + C_1 h f_n + C_2 h \left(f_n + \alpha h f_t'\Big|_n + (\beta h f_n) f_x'\Big|_n\right) + O(h^3)$

and

$x_{n+1} = x_n + f_n h + \frac{1}{2} h^2 (f_t' + f_x' f_n) + O(h^3)$

1) $C_1 + C_2 = 1$

2) $C_2 \alpha = \frac{1}{2}$

3) $C_2 \beta = \frac{1}{2}$

These three equations provide four unknows. These unknowns may lead to an infinite number of possibilities.

45

45

Second-order Runge-Kutta (cont.)

$x_{n+1} = x_n + C_1 h f_n + C_2 h f(x_n + \beta \Delta x_1, t_n + \alpha h)$

1) $C_1 + C_2 = 1$, 2) $C_2 \alpha = \frac{1}{2}$, 3) $C_2 \beta = \frac{1}{2}$

Choice 1: $C_1 = 1, C_2 = 0,$

$x_{n+1} = x_n + h f(x_n, t_n)$ first-order explicit Euler

Choice 2: $C_1 = C_2 = \frac{1}{2}, \alpha = \beta = 1$

$x_{n+1} = x_n + \frac{1}{2} h (f(x_n, t_n) + f(x_n + f_n h, t_n + h))$ the predictor-corrector method

Choice 3: $C_1 = 0, C_2 = 1, \alpha = \beta = \frac{1}{2}$

$x_{n+1} = x_n + h f\left(x_n + \frac{1}{2} f_n h, t_n + \frac{1}{2} h\right)$ the mid-point method

46

46

Second-order Runge-Kutta (cont.)

In general literature, Runge-Kutta formulas frequently denote the Δx_i by k_i

Thus, the second-order Runge-Kutta

$x_{n+1} = x_n + \frac{1}{2} h (f(x_n, t_n) + f(x_n + f_n h, t_n + h))$

is given as

$x_{n+1} = x_n + \frac{1}{2}(k_1 + k_2)$

$k_1 = h f(x_n, t_n) = h f_n$

$k_2 = h f(x_n + k_1, t_n + h)$

(remember $h = \Delta t$)

47

47

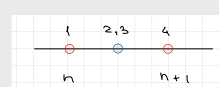
Fourth – order Runge-Kutta

4th order Runge-Kutta is very popular in physics

$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) + O(h^5)$

$k_1 = h f(x_n, t_n)$

$k_2 = h f\left(x_n + \frac{k_1}{2}, t_n + \frac{h}{2}\right)$



$k_3 = h f\left(x_n + \frac{k_2}{2}, t_n + \frac{h}{2}\right)$

$k_4 = h f(x_n + k_3, t_n + h)$

Summary: (characteristics can be explored using equation $x' + \alpha x = 0$)

1. RK4 is explicit and require four derivative function evaluations per step.
2. RK4 is consistent, $O(h^5)$ locally and $O(h^4)$ globally
3. RK4 is conditionally stable
4. Since RK4 is consistent and conditionally stable, thus it's convergent

48

48

Example: C++ RK 4th order

```
/*-----------------------------------------------------------------------
 Program to solve 1st order Ordinary Differential Equations x'(t) = f(t,x)
 method:  4th-order Runge-Kutta method
 input ...
 f(t,x)- function supplied by a user
 ti  - initial value for an independent variable (t)
 xi  - initial value for a function x(t)
 tf  - find solution for this point t
 output ...
 xf  - solution at point tf, i.e. x(tf)
----------------------------------------------------------------------*/
double rk4_1st(double(*f)(double, double), double ti, double xi, double tf)
{
   double xf;
   double h,k1,k2,k3,k4;
   h  = tf-ti;
   k1 = h*f(ti,xi);
   k2 = h*f(ti+h/2.0,xi+k1/2.0);
   k3 = h*f(ti+h/2.0,xi+k2/2.0);
   k4 = h*f(ti+h,xi+k3);
   xf = xi + (k1 + 2.0*(k2+k3) + k4)/6.0;
   return xf;
}
see full code at: https://ww2.odu.edu/~agodunov/book/programs.html
```

49

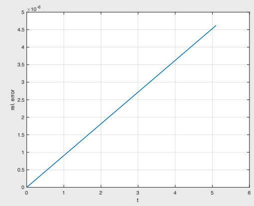
49

Example: relative error Runge-Kutta 4th order

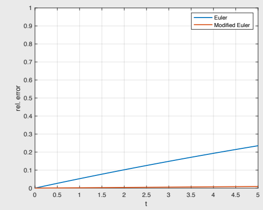
$x'(t) = -x \quad x(t) = e^{-t}, \qquad x(0) = 1;$

$dt = 0.1$

Runge-Kutta



Euler



ATTENTION: note the scale differences – a few orders better for RK

50

50

Error estimation for single-point methods

For a single step

$x_{n+1} = x_{n+1} + Ah^{m+1}$

where x_{n+1} is the exact solution at t_{n+1}, x_{n+1} is the approximate solution and Ah^{m+1} is local truncation error.

Doing calculations for h and twice with $h/2$ steps we can estimate the local truncation error

$$Error = Ah^{m+1} = \frac{2^m}{(2^m - 1)}[x_{n+1}\{h/2\} - x_{\{n+1\}}\{h\}]$$

where $x_{n+1}\{h\}$ is solution with with h step and $x_{n+1}\{h/2\}$ is the solution with two $h/2$ steps.

If $Error < tolerance$ increase (double) the step size. If $Error > tolerance$, decrease (halve) the step size.

This method of error estimation requires 200 percent more work. Consequently, it should be used only occasionally.

51

51

Step-size control

Let us suppose that a step of length h_n has just been completed, and also suppose that it is possible to estimate the local error e_{n+1} of the computed simulation x_{n+1}.

A widely accepted formula for predicting a step-size h_{n+1}, following a successful step of size h_n, is

$$h_{n+1} = 0.9h_n\left[\frac{T}{||e_{n+1}||}\right]^{\frac{1}{p+1}}, \qquad ||e_{n+1}|| \le T$$

If $||e_{n+1}|| > T$, we can use the formula above for estimation, or just reduce the step-size by a factor of 2.

More sophisticated schemes, based on error estimates at two or more successive steps, have been devised for step-size control. These are employed in some computer packages for the solution of differential equations.

52

52

Algorithm

let Tol is desired local tolerance, and $Err = e_{n+1}$ is calculated local error

if Err <= Tol

accept solution

calculate new step as min between the two

a) $h_{n+1} = 0.9h_n\left[\frac{T}{||e_{n+1}||}\right]^{\frac{1}{p+1}}$ where p=4 for RK 4th order

b) $h_{n+1} = 2h_n$ (no more than twice the old step)

if Err > Tol

come back and start from x_n with a smaller step as max between the two

a) $h_{n+1} = 0.9h_n\left[\frac{T}{||e_{n+1}||}\right]^{\frac{1}{p+1}}$

b) $h_{n+1} = 0.5h_n$ (decrease the old step by two)

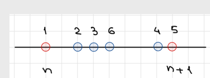
How to calculate Err?

53

53

RKF45 – one of the most popular versions of RK

Runge-Kutta-Fehlberg with error estimation (for step size control)



$k_1 = hf(x_n, t_n)$

$k_2 = hf\left(x_n + \frac{k_1}{4}, t_n + \frac{h}{4}\right)$

$k_3 = hf\left(x_n + \frac{3}{32}k_1 + \frac{9}{32}k_2, t_n + \frac{3}{8}h\right)$

$k_4 = hf\left(x_n + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3, t_n + \frac{12}{13}h\right)$

$k_5 = hf\left(x_n + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4, t_n + h\right)$

$k_6 = hf\left(x_n - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5, t_n + \frac{1}{2}h\right)$

$x_{n+1} = x_n + \left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right)$

$Error = \frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6 + O(h^6)$

54

54

Example: C++ RKF45 (part of a code)

```
while (ti < tmax)
  tf = ti + dt;
  [xf,er5] = RKF51(dx,ti,xi,tf);
  er5 = abs(er5);
  if (er5 <= tolerance)
      i = i + 1;
      x(i) = xf;
      t(i) = tf;
      ti = tf;
      xi = xf;
      dt = min(0.9*dt*(abs(tolerance/er5)).^(1/6),2.0*dt);
    else
      dt = max(0.9*dt*(abs(tolerance/er5)).^(1/6),0.5*dt);
  end
end
```

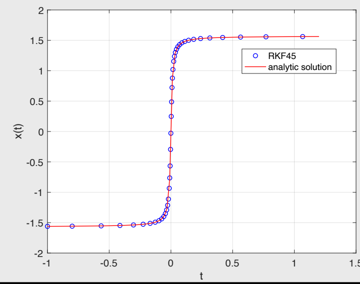
55

55

Example: step-size control

equation

$$\frac{dx}{dt} = \frac{100}{1+10000t^2}, \qquad x(-1) = \arctan(-100), \qquad x(t) = \arctan 100t$$



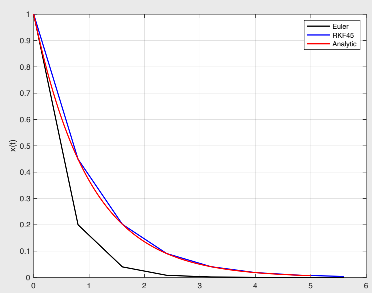
56

56

Example: RKF45 with large step size

$x'(t) = -x \quad x(t) = e^{-t}, \qquad x(0) = 1;$

$dt = 0.8$



57

57

Summary

- Single-point methods work well for both smoothly varying problems and non-smoothly varying problems.
- The first-order Euler methods are useful for illustrating the basic features of finite difference methods for solving initial-value ODEs, but they are too inaccurate to be of any practical value.
- The second-order single-point methods are useful for illustrating techniques for achieving higher-order accuracy, but they also are too inaccurate to be of much practical value.
- Runge-Kutta methods can be developed for any order desired. The fourth-order Runge- Kutta method is the method of choice when a single-point method is desired.
- RKF45 provides error estimation for step-size control

58

58

Part 4:
Additional information

59

Troubles with single-point methods

RK methods introduce intermediate points between n and $n+1$.

High-order RK methods, while successful, require a large number derivative function evaluations, i.e. calls $f(x,t)$.

Higher-order methods requiring fewer derivative function evolutions are desirable.

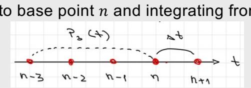
Multipoint methods, which use more than one known point, have this capability.

60

60

Forth-order Adams-Bashforth method

One of the most popular multipoint methods is the fourth order Adams-Bashforth methods, which is obtained by fitting a third-degree Newton backward difference polynomial to base point n and integrating from point n the point $n+1$.



After long and tedious work ….

$$x_{n+1} = x_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3}) + \frac{251}{720}h^5 x^{(5)}(\tau)$$

Consistency and stability analysis for multipoint measures are quite tedious and complicated.

The above method is convergent, conditionally stable (stability is much better than for RK 4th order) and the global error is $O(h^4)$.

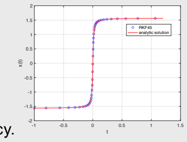
However, error estimation and error control are difficult.

There are very many variations of the general multipoint methods.

61

61

Stiff ODE



Definitions of stiffness

- the step size required for stability is much smaller than the step size required for accuracy.
- if it contains some components of the solution that decay rapidly compared to other components of the solution.
- if the step size based on computational time is too large to obtain an accurate solution.

There is a set of methods developed by Gear (1971) for solving stiff ODEs

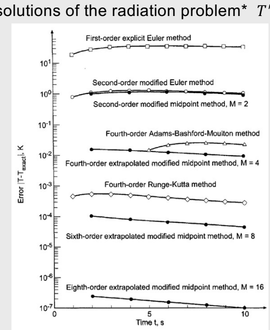
Good package: LSODE developed in Lawrence Livermore National Laboratory (LLNL)

62

62

Summary of methods and results

Errors in solutions of the radiation problem* $T' = -\alpha(T^4 - T_a^4)$, $T(0) = T_0$

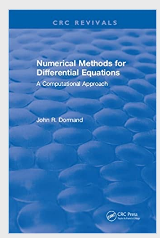


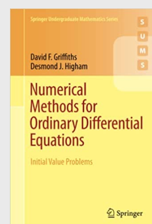
* from Numerical Methods for engineers and scientists by J. Hoffman

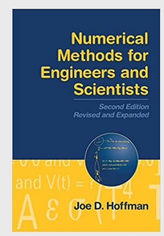
63

63

Books







64

64