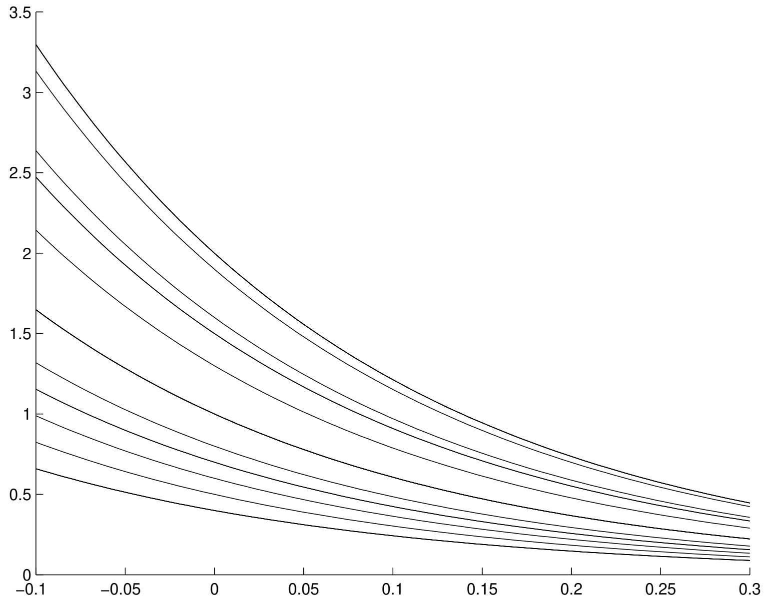
Ordinary Differential Equations

Initial Value Problem (first order): find $y(t)$ such that

$$y' = f(y, t)$$

given initial value $y(t_0)$, usually assume $t_0 = 0$

A differential equation has a family of solutions, each corresponding to an initial value.



- The IVP has a unique solution for any initial value if f is Lipschitz continuous
$$\|f(\hat{y}, t) - f(y, t)\| \leq L\|\hat{y} - y\|.$$
- Traditionally, a solution of an ODE is said to be stable if for any given ϵ there exists a δ such that if $\hat{y}(t)$ is the solution of the ODE with initial value $\hat{y}(t_0)$ and $\|\hat{y}(t_0) - y(t_0)\| \leq \delta$, then $\|\hat{y}(t) - y(t)\| \leq \epsilon$ for all $t \geq t_0$. For example, $y' = \alpha y$. the solution $y(t) = y_0 e^{\alpha t}$ is stable when $\mathrm{Re}(\alpha) \leq 0$ and unstable otherwise. In particular, when $\mathrm{Re}(\alpha) < 0$, all solutions converge (to zero). This is called asymptotically stable.

- Even if a solution is stable, it can still be highly sensitive to the perturbation in the initial value.

Generalization 1: system of first order ODEs: y and f are vectors

Generalization 2: high order

$$u'' = g(u, u', t).$$

Let $y_1 = u$ and $y_2 = u'$ and transform the above into

$$\begin{cases} y_1' = y_2 \\ y_2' = g(y_1, y_2, t) \end{cases}$$

We consider the initial value problem:

$$y' = f(y, t), \qquad y(t_0) = y_0$$

Numerical solution: find approximations

$$y_n \approx y(t_n), \qquad \text{for} \quad t_1, t_2, \ldots$$

Note: $y_0 = y(t_0)$ (initial value)

A k-step method: Compute y_{n+1} using $y_n, y_{n-1}, \ldots, y_{n-k+1}$.

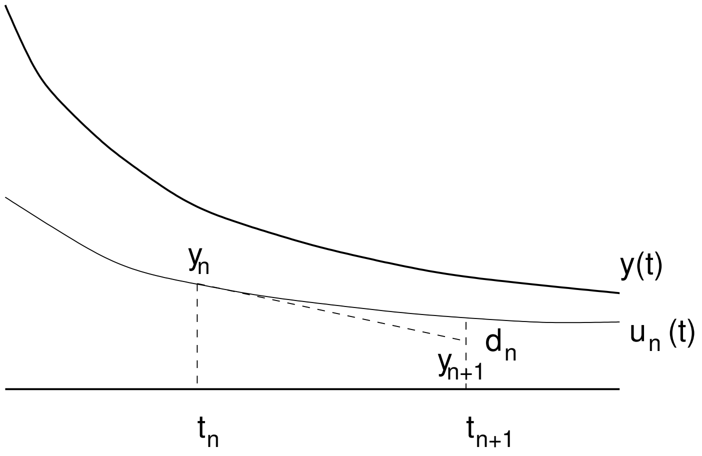
Errors

Two sources of errors: discretization error and roundoff error.

Discretization error: caused by the method used, independent of the computer used and the program implementing the method.

Two types of discretization error

- Global error: $e_n = y_n - y(t_n)$.
- Local error: The error in one step. Consider t_n as the starting point and the approximation y_n at t_n as the initial value , if $u_n(t)$ is the solution of
$$u_n' = f(u_n, t), \quad u_n(t_n) = y_n$$
then the local error is
$$d_n = y_{n+1} - u_n(t_{n+1})$$



Note that y_{n+1} is the approximation at t_{n+1}, $u_n(t)$ is a member of the solution family

$$y' = f(y, t)$$

with $u_n(t_n) = y_n$. Starting with t_0, as we proceed, we jump from one solution in the family to another.

Relation between global error e_n and local error d_n:

- $e_N > \sum_{n=0}^{N-1} d_n$, if the differential equation is unstable.
- $e_N \leq \sum_{n=0}^{N-1} d_n$, if the differential equation is stable.

Stability. A numerical method is said to be stable if small perturbations do not cause the resulting numerical solution to diverge away without bound.

Accuracy. An order p method:

$$|d_n| \leq C h_n^{p+1} \qquad (\text{or } O(h_n^{p+1}))$$

C: independent of n and h_n.

Consider the interval $[t_0, t_N]$ and partition
$t_0, t_1, ..., t_N$.
Roughly, the global error

$$e_N \approx \sum_{n=0}^{N-1} d_n \approx N \cdot O(h^{p+1}) \approx (t_N - t_0) \cdot O(h^p)$$

at the final point t_N is roughly $O(h^p)$ for a method of order p.

For a pth order method, if the subintervals h_n are cut in half, then the average local error is reduced by a factor of 2^{p+1}, the global error is reduced by a factor of 2^p. (But double the number of steps, i.e., more work.)

1 Numerical Differentialtion

The derivative of a function $f(x)$ can be approximated by finite difference:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h_i}, \quad h_i = x_{i+1} - x_i.$$

Finite difference method introduces both discretization error and rounding error.

Automatic Differentiation

Any function expressed by a computer program consists of arithmetic operations and elementary functions, whose derivatives can be easily computed. Thus, the derivative of the function can be computed step by step along with the function by repeated use of the chain rule. For example, $f(x) = (\sin x)^2$, let $y = \sin x$, then $y' = \cos x$ and $(y^2)' = 2y$. Thus, $f'(x) = (2\sin x)(\cos x)$. This is the basic idea behind automatic differentiation. Although the basic idea is simple, its practical implementation is complicated. It requires clever strategies for reducing the potentially explosive complexity. There are effective software packages available. When applicable, automatic differentiation introduces only rounding error, no discretization error.

2 Euler's Method

A single-step method:

$$f(y_0, t_0) = y'(t_0) \approx \frac{y(t_1) - y(t_0)}{h_0},$$

where $h_0 = t_1 - t_0$. The first step:

$$y_1 = y_0 + h_0 f(y_0, t_0)$$

In general,

$$y_{n+1} = y_n + h_n f(y_n, t_n)$$

Local solution $u_n(t)$

$$\begin{aligned} u_n'(t) &= f(u_n(t), t) \\ u_n(t_n) &= y_n. \end{aligned}$$

Taylor expansion at t_n:

$$u_n(t) = u_n(t_n) + (t - t_n)u'(t_n) + O((t - t_n)^2)$$

since $y_n = u_n(t_n)$ and $u'(t_n) = f(y_n, t_n)$, we get

$$u_n(t_{n+1}) = y_n + h_n f(y_n, t_n) + O(h_n^2)$$

Local error

$$d_n = y_{n+1} - u_n(t_{n+1}) = O(h_n^2)$$

Euler's method is a first order method ($p = 1$)

Roundoff error. Each step of Euler's method

$$y_{n+1} = y_n + h_n f(y_n, t_n) + \epsilon \quad |\epsilon| \leq u.$$

Total rounding error: $N\epsilon = b\,\epsilon/h$ ($b = t_N - t_0$, fixed step size h)

$$\text{total error} \approx b\left(Ch + \frac{\epsilon}{h}\right)$$

Remarks

- If h is too small, the roundoff error is large
- If h is too large, the discretization error is large

The total error is minimized by

$$h_{\text{opt}} \approx \sqrt{\frac{u}{C}}$$

recalling that u is the unit of roundoff.

3 Runge-Kutta Methods

Idea: Sample f at several spots to achieve high order.

Cost: More function evaluations

Consider the rectangle rule applied to $y' = f(y,t)$:

$$\begin{aligned} y_{n+1} - y_n &= \int_{t_n}^{t_{n+1}} f(y,t)dt \\ &\approx h_n f(y(t_n + \frac{h_n}{2}), t_n + \frac{h_n}{2}) \\ &\approx h_n f(y_n + \frac{h_n}{2} f(y_n, t_n), t_n + \frac{h_n}{2}), \end{aligned}$$

we get a second-order one-step method

$$\begin{aligned} y_{n+1} &= y_n + k_1, \\ k_1 &= h_n f(y_n + \frac{h_n}{2} f(y_n, t_n), t_n + \frac{h_n}{2}). \end{aligned}$$

This is a second-order Runge-Kutta method.

Second-order Runge-Kutta methods

Suppose

$$y_{n+1} = y_n + \gamma_1 k_0 + \gamma_2 k_1$$

where γ_1 and γ_2 to be determined and

$$\begin{aligned} k_0 &= h_n f(y_n, t_n), \\ k_1 &= h_n f(y_n + \beta k_0, t_n + \alpha h_n), \end{aligned}$$

α and β to be determined. Intuitively, we require $\gamma_1 = 1 - \gamma_2$ and $\alpha = \beta$.

Taylor series (two variables):

$$k_1 = h_n(f_n + \beta k_0 f'_y(y_n, t_n) + \alpha h_n f'_t(y_n, t_n) + \cdots).$$

Thus

$$y_{n+1} = y_n + (\gamma_1 + \gamma_2) h_n f_n + \gamma_2 \beta h_n^2 f_n f'_y(y_n, t_n) + \gamma_2 \alpha h_n^2 f'_t(y_n, t_n) + \cdots$$

The Taylor expansion of the true local solution:

$$\begin{aligned} u_n(t_{n+1}) &= u_n(t_n) + h_n u_n'(t_n) + \frac{h_n^2}{2} u_n''(t_n) + \cdots \\ &= y_n + h_n f_n + \frac{h_n^2}{2}(f_y'(y_n, t_n) f_n + f_t'(y_n, t_n)) + \cdots \end{aligned}$$

Comparing the two expressions, we get

$$\begin{cases} \gamma_1 + \gamma_2 = 1 \\ \gamma_2 \beta = 1/2 \\ \gamma_2 \alpha = 1/2 \end{cases}$$

The second-order Runge-Kutta method

$$y_{n+1} = y_n + \left(1 - \frac{1}{2\alpha}\right) k_0 + \frac{1}{2\alpha} k_1$$

where

$$\begin{aligned} k_0 &= h_n f(y_n, t_n) \\ k_1 &= h_n f(y_n + \alpha k_0, t_n + \alpha h_n). \end{aligned}$$

When $\alpha = 1/2$, related to the rectangle rule.

When $\alpha = 1$, related to the trapezoid rule.

Classical fourth-order Runge-Kutta method

$$y_{n+1} = y_n + \frac{1}{6}(k_0 + 2k_1 + 2k_2 + k_3),$$

where

$$\begin{aligned} k_0 &= hf(y_n, t_n) \\ k_1 &= hf\left(y_n + \frac{1}{2}k_0, t_n + \frac{h}{2}\right) \\ k_2 &= hf\left(y_n + \frac{1}{2}k_1, t_n + \frac{h}{2}\right) \\ k_3 &= hf(y_n + k_2, t_n + h) \end{aligned}$$

4 Multistep Methods

Compute y_{n+1} using $y_n, y_{n-1}, \ldots$ and $f_n, f_{n-1}, \ldots$ possibly f_{n+1} ($f_i = f(y_i, t_i)$).
General linear k-step method:

$$y_{n+1} = \sum_{i=1}^{k} \alpha_i y_{n-i+1} + h \sum_{i=0}^{k} \beta_i f_{n-i+1}.$$

- $\beta_0 = 0$ (no f_{n+1}), explicit method.
- $\beta_0 \neq 0$, implicit method.

Example of multistep methods.
Adams-Bashforth methods:

$$y_{n+1} = y_n + \int_{t_n}^{t_{n+1}} p_{k-1}(t) dt$$

where $p_{k-1}(t)$ is a polynomial of degree $k-1$ which interpolates $f(y,t)$ at (y_{n-j}, t_{n-j}), $j = 0, \ldots, k-1$.
When $p_0(t) = f(y_n, t_n)$, we get Euler method.

Adams-Bashforth family

$y_{n+1} = y_n + hf_n$
local error $\frac{h^2}{2} y^{(2)}(\eta)$, order 1.
$y_{n+1} = y_n + \frac{h}{2}(3f_n - f_{n-1})$
local error $\frac{5h^3}{12} y^{(3)}(\eta)$, order 2.
$y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2})$
local error $\frac{3h^4}{8} y^{(4)}(\eta)$, order 3.
$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$
local error $\frac{251h^5}{720} y^{(5)}(\eta)$, order 4.

The "start-up" issue in multistep methods:
How to get the $k-1$ start values $f_j = f(y_j, t_j))$, $j = 1, \ldots, k-1$? Use a single step method to get start values, then switch to multistep method. (Careful about accuracy consistency.)

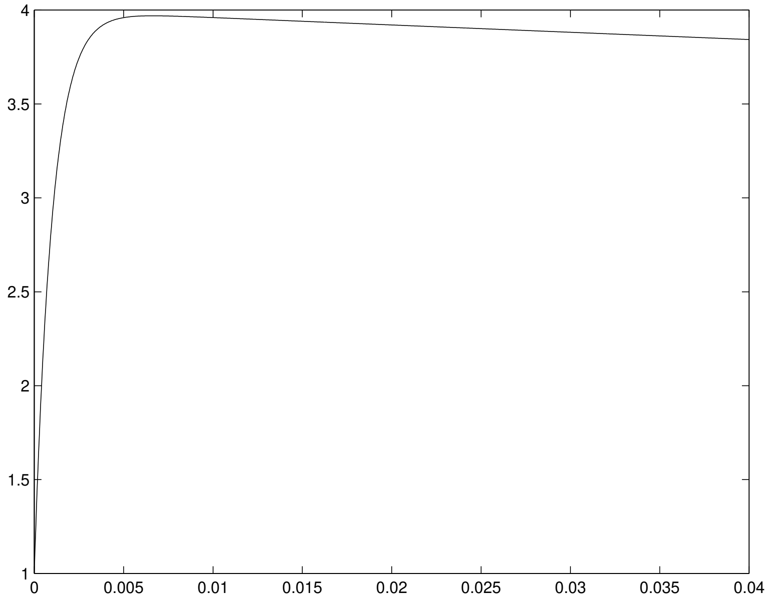


Figure 1: Values of $y_1(t)$.

5 Implicit Methods

Why implicit methods?

Example. A system of two differential equations,

$$\mathbf{y}' = \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix} \mathbf{y}.$$

If the initial values $\mathbf{y}(0) = [1, 1]^{\mathrm{T}}$, then the solution is

$$\mathbf{y} = \begin{bmatrix} 4e^{-t} - 3e^{-1000t} \\ -2e^{-t} + 3e^{-1000t} \end{bmatrix}.$$

Suppose we use Euler's method

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix} \mathbf{y}_n$$

with $\mathbf{y}_0 = [1.0, 1.0]^{\mathrm{T}}$, then we get Table 1.

For comparison, the solution values are given in Table 2

i	h = 0.01	h = 0.001
0	1.0	1.0
1	30.96	3.996
2	-239.1	3.992
3	-9785	-7.988
4	52420	-31.92

Table 1: Values of $y_i(1)$ using Euler's method.

i	h = 0.01	h = 0.001
0	1.0	1.0
1	3.960	2.892
2	3.921	3.586
3	3.882	3.839
4	3.843	3.929

Table 2: Values of $y(t_i)$.

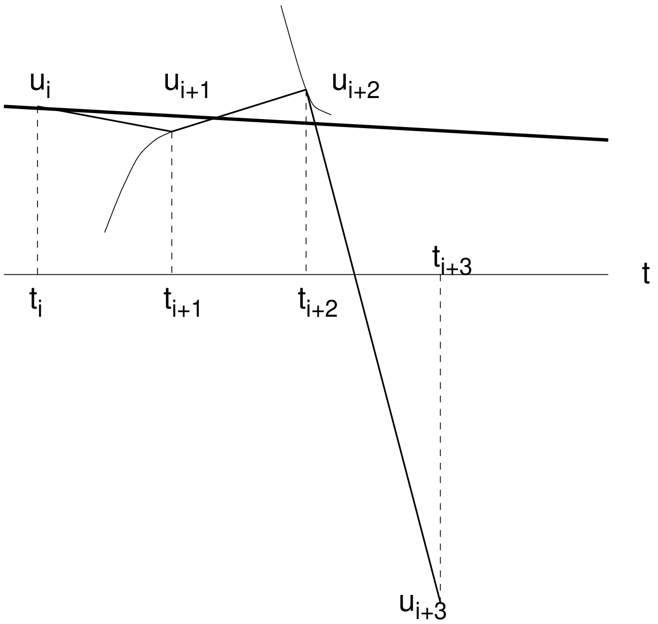


Figure 2: A stiff problem

This kind of ODE is said to be stiff, see Figure 2, where a slowly varying solution is surrounded by solutions with rapidly decaying transients. This is an example of a stable equation but unstable numerical method.

Backward Euler's method. An example of implicit methods.

Taylor expansion of the solution $y(t)$ about $t = t_{n+1}$ (instead of $t = t_n$),

$$\begin{aligned} y(t_{n+1} + h) &\approx y(t_{n+1}) + y'(t_{n+1})h \\ &= y(t_{n+1}) + f(y(t_{n+1}), t_{n+1})h \end{aligned}$$

and set $h = -h_n = (t_n - t_{n+1})$, then we get

$$y(t_n) \approx y(t_{n+1}) - h_n f(y(t_{n+1}), t_{n+1}).$$

Substituting y_n for $y(t_n)$ and y_{n+1} for $y(t_{n+1})$, we have the backward Euler method

$$y_{n+1} = y_n + h_n f(y_{n+1}, t_{n+1}).$$

Implicit methods tend to be more stable than their explicit counterparts. But there is a price, y_{n+1} is a zero of a nonlinear function.

If we apply the backward Euler's method

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \begin{bmatrix} 998 & 1998 \\ -999 & -1999 \end{bmatrix} \mathbf{y}_{n+1}$$

with $\mathbf{y}_0 = [1.0, 1.0]^{\mathrm{T}}$, then we get Table 3

Trapezoidal method. Integrating $y' = f(y, t)$ over $[t_n, t_{n+1}]$, we get

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(y, t)dt.$$

- Using $f(y_n, t_n) \approx f(y, t)$, we get Euler's method.
- Using $f(y_{n+1}, t_{n+1}) \approx f(y, t)$, we get backward Euler's method.
- Using $f(y_n + \frac{h_n}{2} f_n, t_n + \frac{h_n}{2}) \approx f(y, t)$, we get the second order Runge-Kutta method.

i	h = 0.01	h = 0.001
0	1.0	1.0
1	3.688	2.496
2	3.896	3.242
3	3.880	3.613
4	3.844	3.797

Table 3: Values of $y_i(1)$ using backward Euler's method.

Applying the trapezoidal quadrature rule to the integration, we have

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(y(t_n), t_n) + f(y(t_{n+1}), t_{n+1})) + O(h^3)$$

and a second-order one-step implicit method

$$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1}).$$

How to find y_{n+1}? Usually, linear iteration

$$y_{n+1}^{(j+1)} = y_n + \frac{h}{2}(f_n + f(y_{n+1}^{(j)}, t_{n+1}))$$

is sufficient. Since

$$y_{n+1} - y_{n+1}^{(j+1)} = \frac{h}{2}(f(y_{n+1}, t_{n+1}) - f(y_{n+1}^{(j)}, t_{n+1})),$$

the Lipschitz condition implies that

$$|y_{n+1} - y_{n+1}^{(j+1)}| \le \frac{hL}{2}|y_{n+1} - y_{n+1}^{(j)}|.$$

Thus, $y_{n+1}^{(j)}$ converges to y_{n+1} if

$$\frac{hL}{2} < 1.$$

The initial $y_{n+1}^{(0)}$ can be obtained by an explicit method.

Consistency. Since the local error is $O(h^3)$, we must choose the initial $y_{n+1}^{(0)}$ and iterate enough times so that the iteration error $|y_{n+1}^{(j)} - y_{n+1}| = O(h^3)$. Usually, we want the iteration error to be less significant than the local error and require

$$|y_{n+1}^{(j)} - y_{n+1}| = O(h^4)$$

for the trapezoidal method.

Combining both explicit and implicit: Use an explicit method for predictor and an implicit method for corrector.

Example. Fourth-order Adam's method.
Predictor (fourth-order, explicit):

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$

Corrector (fourth-order, implicit):

$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$$

Algorithm. Predict-correct.

1. Prediction: use the predictor to calculate $y_{n+1}^{(0)}$
2. Evaluation: $f_{n+1}^{(0)} = f(y_{n+1}^{(0)}, t_{n+1})$
3. Correction: use the corrector to calculate $y_{n+1}^{(1)}$

Steps 2 and 3 are repeated until $|y_{n+1}^{(i+1)} - y_{n+1}^{(i)}| \leq tol$. Then set $y_{n+1} = y_{n+1}^{(i+1)}$.

Usually, PECE, one iteration and a final evaluation for the next step.

6 Boundary Value Problem

Second order equation

$$y''(t) = f(y(t), y'(t), t)$$

with boundary values

$$y(0) = \alpha \quad \text{and} \quad y(1) = \beta.$$

Let $u(t) = y(t)$ and $v(t) = y'(t)$, then the second order equation can be transformed into a system of two first order equations:

$$\begin{aligned} u'(t) &= v(t) \\ v'(t) &= f(u(t), v(t), t). \end{aligned}$$

We know one initial value $u(0) = y(0) = \alpha$, but we do not know the other initial value $v(0) = y'(0)$. Instead we know a boundary value $y(1) = u(1)$.

Shooting method

The idea behind shooting method is:

1. make a guess $z = v(0)$;
2. now we have $u(0)$ and $v(0)$, solve the initial value problem;
3. if $u_N \neq \beta$, $t_N = 1$, adjust the guess z.

Thus, every guess $z = v(0)$ corresponds to a u_N. We can define a function ϕ: $z \rightarrow u_N - \beta$. The initial value $v(1) = y'(1)$ is a zero of the function ϕ. It then becomes a zero finding problem. For example, we can use linear interpolation to find a zero:

1. make two guesses z_1 and z_2;
2. solve the initial value problems, get $\phi(z_1)$ and $\phi(z_2)$;
3. linearly interpolate $(\phi(z_1), z_1)$, $(\phi(z_2), z_2)$ (inverse linear interpolation) and find z_3 so that $\phi(z_3) = 0$;
4. repeat until $\phi(z_k) \approx 0$.

Alternatively, the inverse linear interpolation can be replaced by the inverse cubic interpolation. Once we find $v(1)$, we can find the numerical solution $u_N = y_N$ by solving the initial value problme.

Finite difference method

Partition the interval $[0, 1]$:

$$t_i = ih, \quad h = \frac{1}{N}, \quad i = 0, ..., N.$$

Approximate $y'(t)$ and $y''(t)$ by the finite differences:

$$\frac{1}{2h}[y(t+h) - y(t-h)]$$

and

$$\frac{1}{h^2}[y(t+h) - 2y(t) + y(t-h)]$$

respectively. On each subinterval $[t_i, t_{i+2}]$, $i = 0, ..., N-2$, the second order differential equation $y''(t) = f(y(t), y'(t), t)$ is approximated by a linear/nonlinear equation. The numerical solution $y_1, y_2, ..., y_{N-1}$ can be computed by solving a system of $N-1$ linear/nonlinear equations.

Summary

- Family of solutions of a differential equation, initial value problem.
- Transform a high order ODE into a system of first order ODEs.
- Errors: Discretization errors (global, local), stability of a differential equation (mathematical stability), stability of a numerical methos, roundoff error.
- Accuracy: Order of a method.
- Euler's method: Explicit, single step, first order.
- Runge-Kutta method: Explicit, single step.
- Multistep methods: Adams-Bashforth family.
- Implicit methods: Backward Euler method.
- Prediction-Correction scheme: Combination of explicit and implicit methods.
- Boundary value problem: shooting method, finite difference method.

Software packages

IMSL ivprk (RK), ivpag (AB)

MATLAB ode23, ode45 (RK), ode113 (AB), ode15s, ode23s (stiff), bvp4c (BVP)

NAG d02baf (RK), d02caf (AB), d02eaf (stiff)

Netlib dverk (RK), ode (AB), vode, vodpk (sfiff)

References

George E. Forsythe, Michael A. Malcolm, and Cleve B. Moler, *Computer Methods for Mathematical Computations*, Prentice-Hall, Inc., 1977.

Michael T. Heath, *Scientific Computing: An Introductory Survey*, 2nd Edition, McGraw-Hill, 2002.