Euler method

In mathematics and computational science, the **Euler method** (also called **forward Euler method**) is a first-order numerical procedure for solving ordinary differential equations (ODEs) with a given initial value. It is the most basic explicit method for numerical integration of ordinary differential equations and is the simplest Runge–Kutta method. The Euler method is named after Leonhard Euler, who treated it in his book *Institutionum calculi integralis* (published 1768–1870).[1]

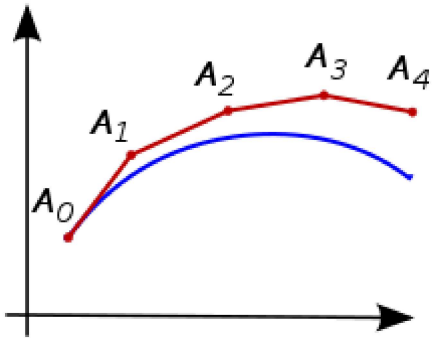


Illustration of the Euler method. The unknown curve is in blue, and its polygonal approximation is in red.

The Euler method is a first-order method, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size. The Euler method often serves as the basis to construct more complex methods, e.g., predictor–corrector method.

Contents

Informal geometrical description

Consider the problem of calculating the shape of an unknown curve which starts at a given point and satisfies a given differential equation. Here, a differential equation can be thought of as a formula by which the slope of the tangent line to the curve can be computed at any point on the curve, once the position of that point has been calculated.

The idea is that while the curve is initially unknown, its starting point, which we denote by A_0, is known (see the picture on top right). Then, from the differential equation, the slope to the curve at A_0 can be computed, and so, the tangent line.

Take a small step along that tangent line up to a point A_1. Along this small step, the slope does not change too much, so A_1 will be close to the curve. If we pretend that A_1 is still on the curve, the same reasoning as for the point A_0 above can be used. After several steps, a polygonal curve $A_0 A_1 A_2 A_3 \dots$ is computed. In general, this curve does not diverge too far from the original unknown curve, and the error between the two curves can be made small if the step size is small enough and the interval of computation is finite:[2]

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

Choose a value h for the size of every step and set $t_n = t_0 + nh$. Now, one step of the Euler method from t_n to $t_{n+1} = t_n + h$ is:[3]

$$y_{n+1} = y_n + hf(t_n, y_n).$$

The value of y_n is an approximation of the solution to the ODE at time t_n: $y_n \approx y(t_n)$. The Euler method is explicit, i.e. the solution y_{n+1} is an explicit function of y_i for $i \leq n$.

While the Euler method integrates a first-order ODE, any ODE of order N can be represented as a system of first-order ODEs: to treat the equation

$$y^{(N)}(t) = f(t, y(t), y'(t), \dots, y^{(N-1)}(t)),$$

we introduce auxiliary variables $z_1(t) = y(t), z_2(t) = y'(t), \dots, z_N(t) = y^{(N-1)}(t)$ and obtain the equivalent equation:

$$\mathbf{z}'(t) = \begin{pmatrix} z_1'(t) \\ \vdots \\ z_{N-1}'(t) \\ z_N'(t) \end{pmatrix} = \begin{pmatrix} y'(t) \\ \vdots \\ y^{(N-1)}(t) \\ y^{(N)}(t) \end{pmatrix} = \begin{pmatrix} z_2(t) \\ \vdots \\ z_N(t) \\ f(t, z_1(t), \dots, z_N(t)) \end{pmatrix}$$

This is a first-order system in the variable $\mathbf{z}(t)$ and can be handled by Euler's method or, in fact, by any other scheme for first-order systems.[4]

Example

Given the initial value problem

$$y' = y, \quad y(0) = 1,$$

we would like to use the Euler method to approximate $y(4)$.[5]

Using step size equal to 1 ($h = 1$)

The Euler method is

$$y_{n+1} = y_n + hf(t_n, y_n).$$

so first we must compute $f(t_0, y_0)$. In this simple differential equation, the function f is defined by $f(t, y) = y$. We have

$$f(t_0, y_0) = f(0, 1) = 1.$$

By doing the above step, we have found the slope of the line that is tangent to the solution curve at the point $(0, 1)$. Recall that the slope is defined as the change in y divided by the change in t, or $\Delta y/\Delta t$.

The next step is to multiply the above value by the step size h, which we take equal to one here:

$$h \cdot f(y_0) = 1 \cdot 1 = 1.$$

Since the step size is the change in t, when we multiply the step size and the slope of the tangent, we get a change in y value. This value is then added to the initial y value to obtain the next value to be used for computations.

$$y_0 + hf(y_0) = y_1 = 1 + 1 \cdot 1 = 2.$$

The above steps should be repeated to find y_2, y_3 and y_4.

$$y_2 = y_1 + hf(y_1) = 2 + 1 \cdot 2 = 4,$$
$$y_3 = y_2 + hf(y_2) = 4 + 1 \cdot 4 = 8,$$
$$y_4 = y_3 + hf(y_3) = 8 + 1 \cdot 8 = 16.$$

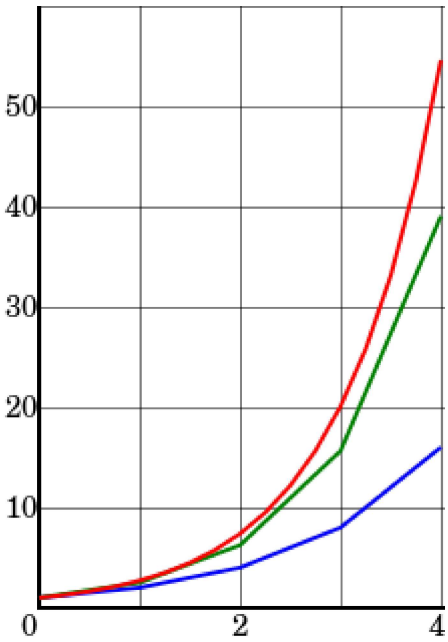


Illustration of numerical integration for the equation $y' = y, y(0) = 1$. Blue is the Euler method; green, the midpoint method; red, the exact solution, $y = e^t$. The step size is $h = 1.0$.

Due to the repetitive nature of this algorithm, it can be helpful to organize computations in a chart form, as seen below, to avoid making errors.

n	y_n	t_n	$f(t_n, y_n)$	h	Δy	y_{n+1}
0	1	0	1	1	1	2
1	2	1	2	1	2	4
2	4	2	4	1	4	8
3	8	3	8	1	8	16

The conclusion of this computation is that $y_4 = 16$. The exact solution of the differential equation is $y(t) = e^t$, so $y(4) = e^4 \approx 54.598$. Although the approximation of the Euler method was not very precise in this specific case, particularly due to a large value step size h, its behaviour is qualitatively correct as the figure shows.

MATLAB code example

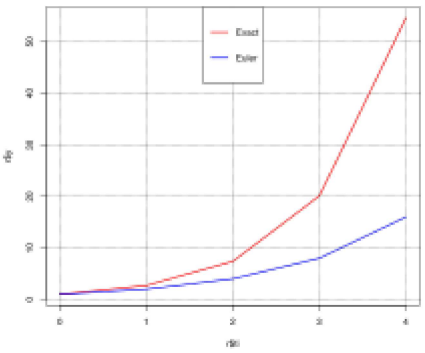
```
clear; clc; close('all');
y0 = 1;
t0 = 0;
h = 1; % try: h = 0.01
tn = 4; % equal to: t0 + h*n, with n the number of steps
[t, y] = Euler(t0, y0, h, tn);
plot(t, y, 'b');

    % exact solution (y = e^t):
    tt = (t0:0.001:tn)';
    yy = exp(tt);
    hold('on');
    plot(tt, yy, 'r');
    hold('off');
    legend('Euler', 'Exact');

function [t, y] = Euler(t0, y0, h, tn)
    fprintf('%10s%10s%10s%15s\n', 'i', 'yi', 'ti', 'f(yi,ti)');
    fprintf('%10d%+10.2f%+10.2f%+15.2f\n', 0, y0, t0, f(y0,t0));
    t = (t0:h:tn)';
    y = zeros(size(t));
    y(1) = y0;
    for i = 1:1:length(t)-1
        y(i+1) = y(i) + h*f(y(i),t(i));
        fprintf('%10d%+10.2f%+10.2f%+15.2f\n', i, y(i+1), t(i+1), f(y(i+1),t(i+1)));
    end
end

% in this case, f(y,t) = f(y)
function dydt = f(y,t)
    dydt = y;
end

% OUTPUT:
%         i        yi        ti       f(yi,ti)
%         0     +1.00     +0.00          +1.00
%         1     +2.00     +1.00          +2.00
%         2     +4.00     +2.00          +4.00
%         3     +8.00     +3.00          +8.00
%         4    +16.00     +4.00         +16.00
% NOTE: Code also outputs a comparison plot
```

R code example

Below is the code of the example in the R programming language.

```
# ============
# SOLUTION to
#   y' = y,   where y' = f(t,y)
# then:
f <- function(ti,y) y

# INITIAL VALUES:
t0 <- 0
y0 <- 1
h  <- 1
tn <- 4
```

```r
# Euler's method: function definition
Euler <- function(t0, y0, h, tn, dy.dt) {
  # dy.dt: derivative function

  # t sequence:
  tt <- seq(t0, tn, by=h)
  # table with as many rows as tt elements:
  tbl <- data.frame(ti=tt)
  tbl$yi <- y0 # Initializes yi with y0
  tbl$Dy.dt[1] <- dy.dt(tbl$ti[1],y0) # derivative
  for (i in 2:nrow(tbl)) {
    tbl$yi[i] <- tbl$yi[i-1] + h*tbl$Dy.dt[i-1]
    # For next iteration:
    tbl$Dy.dt[i] <- dy.dt(tbl$ti[i],tbl$yi[i])
  }
  return(tbl)
}

# Euler's method: function application
r <- Euler(t0, y0, h, tn, f)
rownames(r) <- 0:(nrow(r)-1) # to coincide with index n

# Exact solution for this case: y = exp(t)
#       added as an additional column to r
r$y <- exp(r$ti)

# TABLE with results:
print(r)

plot(r$ti, r$y, type="l", col="red", lwd=2)
lines(r$ti, r$yi, col="blue", lwd=2)
grid(col="black")
legend("top",  legend = c("Exact", "Euler"), lwd=2,  col = c("red",
"blue"))

# OUTPUT:
#
#   ti yi Dy.dt         y
# 0  0  1     1  1.000000
# 1  1  2     2  2.718282
# 2  2  4     4  7.389056
# 3  3  8     8 20.085537
# 4  4 16    16 54.598150
# NOTE: Code also outputs a comparison plot
```



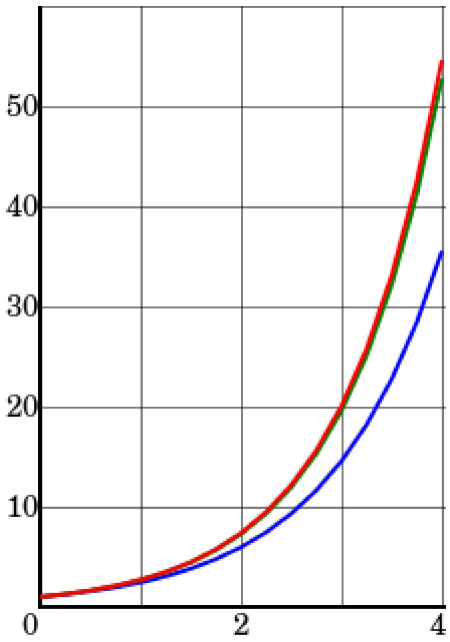
Graphical output of the R programming language code for the posed example

Using other step sizes

As suggested in the introduction, the Euler method is more accurate if the step size h is smaller. The table below shows the result with different step sizes. The top row corresponds to the example in the previous section, and the second row is illustrated in the figure.

step size	result of Euler's method	error
1	16.00	38.60
0.25	35.53	19.07
0.1	45.26	9.34
0.05	49.56	5.04
0.025	51.98	2.62
0.0125	53.26	1.34

The error recorded in the last column of the table is the difference between the exact solution at $t = 4$ and the Euler approximation. In the bottom of the table, the step size is half the step size in the previous row, and the error is also approximately half the error in the previous row. This suggests that the error is roughly proportional to the step size, at least for fairly small values of the step size. This is true in general, also for other equations; see the section *Global truncation error* for more details.



The same illustration for h = 0.25.

Other methods, such as the midpoint method also illustrated in the figures, behave more favourably: the global error of the midpoint method is roughly proportional to the *square* of the step size. For this reason, the Euler method is said to be a first-order method, while the midpoint method is second order.

We can extrapolate from the above table that the step size needed to get an answer that is correct to three decimal places is approximately 0.00001, meaning that we need 400,000 steps. This large number of steps entails a high computational cost. For this reason, people usually employ alternative, higher-order methods such as Runge–Kutta methods or linear multistep methods, especially if a high accuracy is desired.[6]

Derivation

The Euler method can be derived in a number of ways. Firstly, there is the geometrical description above.

Another possibility is to consider the Taylor expansion of the function y around t_0:

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2y''(t_0) + O(h^3).$$

The differential equation states that $y' = f(t, y)$. If this is substituted in the Taylor expansion and the quadratic and higher-order terms are ignored, the Euler method arises.[7] The Taylor expansion is used below to analyze the error committed by the Euler method, and it can be extended to produce Runge–Kutta methods.

A closely related derivation is to substitute the forward finite difference formula for the derivative,

$$y'(t_0) \approx \frac{y(t_0 + h) - y(t_0)}{h}$$

in the differential equation $y' = f(t, y)$. Again, this yields the Euler method.[8] A similar computation leads to the midpoint method and the backward Euler method.

Finally, one can integrate the differential equation from t_0 to $t_0 + h$ and apply the fundamental theorem of calculus to get:

$$y(t_0+h)-y(t_0)=\int_{t_0}^{t_0+h} f(t,y(t))\,\mathrm{d}t.$$

Now approximate the integral by the left-hand rectangle method (with only one rectangle):

$$\int_{t_0}^{t_0+h} f(t,y(t))\,\mathrm{d}t \approx hf(t_0,y(t_0)).$$

Combining both equations, one finds again the Euler method.[9] This line of thought can be continued to arrive at various linear multistep methods.

Local truncation error

The local truncation error of the Euler method is the error made in a single step. It is the difference between the numerical solution after one step, y_1, and the exact solution at time $t_1 = t_0 + h$. The numerical solution is given by

$$y_1 = y_0 + hf(t_0, y_0).$$

For the exact solution, we use the Taylor expansion mentioned in the section *Derivation* above:

$$y(t_0+h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2y''(t_0) + O(h^3).$$

The local truncation error (LTE) introduced by the Euler method is given by the difference between these equations:

$$\mathrm{LTE} = y(t_0+h) - y_1 = \frac{1}{2}h^2y''(t_0) + O(h^3).$$

This result is valid if y has a bounded third derivative.[10]

This shows that for small h, the local truncation error is approximately proportional to h^2. This makes the Euler method less accurate (for small h) than other higher-order techniques such as Runge-Kutta methods and linear multistep methods, for which the local truncation error is proportional to a higher power of the step size.

A slightly different formulation for the local truncation error can be obtained by using the Lagrange form for the remainder term in Taylor's theorem. If y has a continuous second derivative, then there exists a $\xi \in [t_0, t_0+h]$ such that

$$\mathrm{LTE} = y(t_0+h) - y_1 = \frac{1}{2}h^2y''(\xi).$$[11]

In the above expressions for the error, the second derivative of the unknown exact solution y can be replaced by an expression involving the right-hand side of the differential equation. Indeed, it follows from the equation $y' = f(t,y)$ that

$$y''(t_0) = \frac{\partial f}{\partial t}(t_0, y(t_0)) + \frac{\partial f}{\partial y}(t_0, y(t_0))\, f(t_0, y(t_0)).$$[12]

Global truncation error

The global truncation error is the error at a fixed time t, after however many steps the methods needs to take to reach that time from the initial time. The global truncation error is the cumulative effect of the local truncation errors committed in each step.[13] The number of steps is easily determined to be $(t - t_0)/h$, which is proportional to $1/h$, and the error committed in each step is proportional to h^2 (see the previous section). Thus, it is to be expected that the global truncation error will be proportional to h.[14]

This intuitive reasoning can be made precise. If the solution y has a bounded second derivative and f is Lipschitz continuous in its second argument, then the global truncation error (GTE) is bounded by

$$|\mathrm{GTE}| \leq \frac{hM}{2L}(e^{L(t-t_0)} - 1)$$

where M is an upper bound on the second derivative of y on the given interval and L is the Lipschitz constant of f.[15]

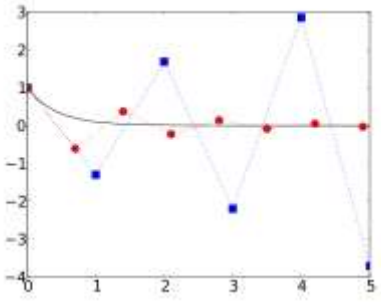
The precise form of this bound is of little practical importance, as in most cases the bound vastly overestimates the actual error committed by the Euler method.[16] What is important is that it shows that the global truncation error is (approximately) proportional to h. For this reason, the Euler method is said to be first order.[17]

Numerical stability

The Euler method can also be numerically unstable, especially for stiff equations, meaning that the numerical solution grows very large for equations where the exact solution does not. This can be illustrated using the linear equation

$$y' = -2.3y, \qquad y(0) = 1.$$

The exact solution is $y(t) = e^{-2.3t}$, which decays to zero as $t \to \infty$. However, if the Euler method is applied to this equation with step size $h = 1$, then the numerical solution is qualitatively wrong: It oscillates and grows (see the figure). This is what it means to be unstable. If a smaller step size is used, for instance $h = 0.7$, then the numerical solution does decay to zero.

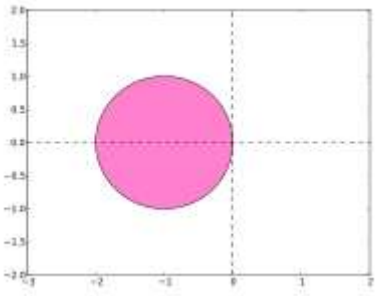


Solution of $y' = -2.3y$ computed with the Euler method with step size $h = 1$ (blue squares) and $h = 0.7$ (red circles). The black curve shows the exact solution.

If the Euler method is applied to the linear equation $y' = ky$, then the numerical solution is unstable if the product hk is outside the region

$$\{z \in \mathbf{C} \mid |z + 1| \leq 1\},$$

illustrated on the right. This region is called the (linear) *stability region*.[18] In the example, k is −2.3, so if $h = 1$ then $hk = -2.3$ which is outside the stability region, and thus the numerical solution is unstable.



The pink disk shows the stability region for the Euler method.

This limitation —along with its slow convergence of error with h— means that the Euler method is not often used, except as a simple example of numerical integration.

Rounding errors

The discussion up to now has ignored the consequences of rounding error. In step n of the Euler method, the rounding error is roughly of the magnitude εy_n where ε is the machine epsilon. Assuming that the rounding errors are all of approximately the same size, the combined rounding error in N steps is roughly $N\varepsilon y_0$ if all errors points in the same direction. Since the number of steps is inversely proportional to the step size h, the total rounding error is proportional to ε / h. In reality, however, it is extremely unlikely that all rounding errors point in the same direction. If instead it is assumed that the rounding errors are independent random variables, then the expected total rounding error is proportional to $\varepsilon/\sqrt{h}$.[19]

Thus, for extremely small values of the step size, the truncation error will be small but the effect of rounding error may be big. Most of the effect of rounding error can be easily avoided if compensated summation is used in the formula for the Euler method.[20]

Modifications and extensions

A simple modification of the Euler method which eliminates the stability problems noted in the previous section is the backward Euler method:

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

This differs from the (standard, or forward) Euler method in that the function f is evaluated at the end point of the step, instead of the starting point. The backward Euler method is an implicit method, meaning that the formula for the backward Euler method has y_{n+1} on both sides, so when applying the backward Euler method we have to solve an equation. This makes the implementation more costly.

Other modifications of the Euler method that help with stability yield the exponential Euler method or the semi-implicit Euler method.

More complicated methods can achieve a higher order (and more accuracy). One possibility is to use more function evaluations. This is illustrated by the midpoint method which is already mentioned in this article:

$$y_{n+1} = y_n + hf\left(t_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hf(t_n, y_n)\right).$$

This leads to the family of Runge–Kutta methods.

The other possibility is to use more past values, as illustrated by the two-step Adams–Bashforth method:

$$y_{n+1} = y_n + \tfrac{3}{2}hf(t_n, y_n) - \tfrac{1}{2}hf(t_{n-1}, y_{n-1}).$$

This leads to the family of linear multistep methods. There are other modifications which uses techniques from compressive sensing to minimize memory usage[21]

In popular culture

In the film *Hidden Figures*, Katherine Goble resorts to the Euler method in calculating the re-entry of astronaut John Glenn from Earth orbit.[22]

See also

- Crank–Nicolson method
- Dynamic errors of numerical methods of ODE discretization
- Gradient descent similarly uses finite steps, here to find minima of functions
- List of Runge-Kutta methods
- Linear multistep method
- Numerical integration (for calculating definite integrals)
- Numerical methods for ordinary differential equations

Notes

1. Butcher 2003, p. 45; Hairer, Nørsett & Wanner 1993, p. 35
2. Atkinson 1989, p. 342; Butcher 2003, p. 60
3. Butcher 2003, p. 45; Hairer, Nørsett & Wanner 1993, p. 36
4. Butcher 2003, p. 3; Hairer, Nørsett & Wanner 1993, p. 2
5. See also Atkinson 1989, p. 344
6. Hairer, Nørsett & Wanner 1993, p. 40
7. Atkinson 1989, p. 342; Hairer, Nørsett & Wanner 1993, p. 36
8. Atkinson 1989, p. 342
9. Atkinson 1989, p. 343
10. Butcher 2003, p. 60
11. Atkinson 1989, p. 342
12. Stoer & Bulirsch 2002, p. 474
13. Atkinson 1989, p. 344
14. Butcher 2003, p. 49
15. Atkinson 1989, p. 346; Lakoba 2012, equation (1.16)
16. Iserles 1996, p. 7
17. Butcher 2003, p. 63
18. Butcher 2003, p. 70; Iserles 1996, p. 57
19. Butcher 2003, pp. 74–75
20. Butcher 2003, pp. 75–78

21. Unni, M. P.; Chandra, M. G.; Kumar, A. A. (March 2017). "Memory reduction for numerical solution of differential equations using compressive sensing" (https://ieeexplore.ieee.org/document/8064928/). *2017 IEEE 13th International Colloquium on Signal Processing its Applications (CSPA)*: 79–84. doi:10.1109/CSPA.2017.8064928 (https://doi.org/10.1109%2FCSPA.2017.8064928).
22. Khan, Amina. "Meet the 'Hidden Figures' mathematician who helped send Americans into space" (http://www.latimes.com/science/sciencenow/la-sci-sn-hidden-figures-katherine-johnson-20170109-story.html). *Los Angeles Times*. Retrieved 12 February 2017.

References

- Atkinson, Kendall A. (1989). *An Introduction to Numerical Analysis* (2nd ed.). New York: John Wiley & Sons. ISBN 978-0-471-50023-0.
- Ascher, Uri M.; Petzold, Linda R. (1998). *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations* (https://books.google.com/books?id=VL51G5JYYAYC&printsec=frontcover&dq=isbn:9780898714128&hl=en&sa=X&ved=0ahUKEwj3sNDO7q3jAhXrAp0JHZykA7cQ6AEIKjAA#v=onepage&q=Euler&f=false). Philadelphia: Society for Industrial and Applied Mathematics. ISBN 978-0-89871-412-8.
- Butcher, John C. (2003). *Numerical Methods for Ordinary Differential Equations*. New York: John Wiley & Sons. ISBN 978-0-471-96758-3.
- Hairer, Ernst; Nørsett, Syvert Paul; Wanner, Gerhard (1993). *Solving ordinary differential equations I: Nonstiff problems*. Berlin, New York: Springer-Verlag. ISBN 978-3-540-56670-0.
- Iserles, Arieh (1996). *A First Course in the Numerical Analysis of Differential Equations* (https://books.google.com/books?id=7Zofw3SFTWIC&printsec=frontcover#v=snippet&q=%22Euler%20method%22&f=false). Cambridge University Press. ISBN 978-0-521-55655-2.
- Stoer, Josef; Bulirsch, Roland (2002). *Introduction to Numerical Analysis* (3rd ed.). Berlin, New York: Springer-Verlag. ISBN 978-0-387-95452-3.
- Lakoba, Taras I. (2012), *Simple Euler method and its modifications* (http://www.cems.uvm.edu/~lakobati/math337/notes_1.pdf) (PDF) (Lecture notes for MATH334), University of Vermont, retrieved 29 February 2012
- Unni, M P. (2017). *Memory reduction for numerical solution of differential equations using compressive sensing* (https://ieeexplore.ieee.org/document/8064928/). IEEE CSPA. ISBN 978-1-5090-1184-1.

External links

- Media related to Euler method at Wikimedia Commons
- Euler method implementations in different languages (http://rosettacode.org/wiki/Euler_method) by Rosetta Code
- Hazewinkel, Michiel, ed. (2001) [1994], "Euler method" (https://www.encyclopediaofmath.org/index.php?title=p/e036530), *Encyclopedia of Mathematics*, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4