

Ordinary differential equations (part 3)

Outline:

- multistep methods (1-st order ODE)
 - Adams-Bashforth
 - Adams-Moulton
 - Predictor-Corrector
 - Backward Differentiation Formulae
- time step control
 - Richardson extrapolation
 - step doubling
 - error control
- multiscale problems (stiffness)
 - implicit methods
- methods for 2-nd order ODE
 - Verlet: standard, position and velocity versions

- Yet we have learnt about two high-order class of methods: RK family and symplectic integrators.
- Both share the common feature: the higher accuracy is requested the more calls to r.h.s. is needed what limits their efficiency.

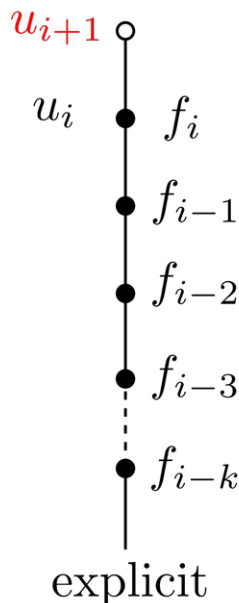
Remark: In most cases the largest part of work in solving ODE comes from calculation of derivative.

- Now one may ask:
 - If there are other methods which do not have this drawback?
 - How can we avoid an issue involving several calls of r.h.s. of ODE?

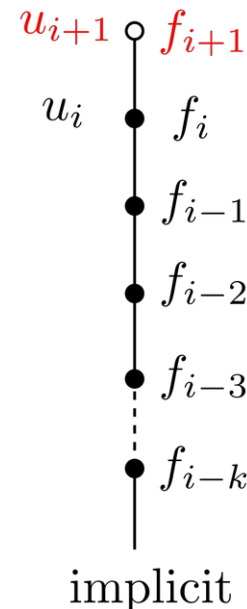
An attempt to answer to these question brings to our attention, that the multistage approach in single time step which gather needed information about first derivative must be replaced with informations coming from the past. This brings us to the concept of **linear multistep methods**. In these methods the already computed values of f (r.h.s. of ODE) at previous time steps are reused in finding the high order solution in the next step

red color denotes
the quantities
to be evaluated

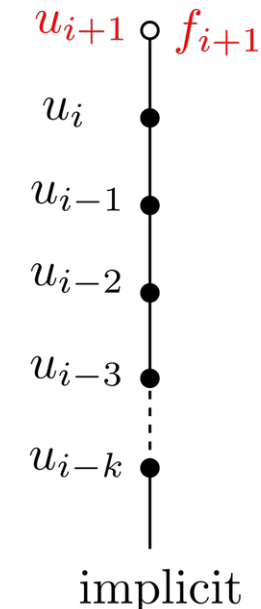
Adams
Bashforth



Adams
Moulton



Backward
Differences



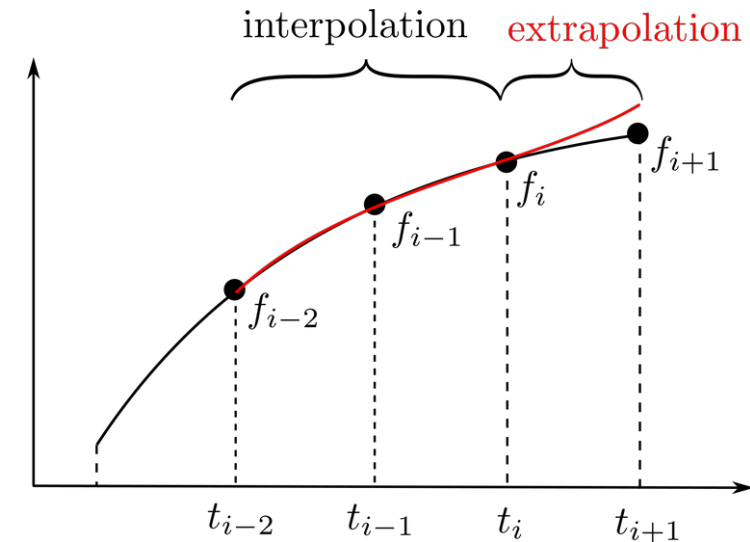
Adams-Bashforth method (explicit)

We know that ODE can be transformed into integral equation

$$\frac{du}{dt} = f(t, u) \quad \Rightarrow \quad u_{t_i+\Delta t} = u(t_i) + \int_{t_i}^{t_i+\Delta t} dt f(t, u)$$

Solving this integral is problematic since we don't know the value of $f(t, u)$. However we may **extrapolate** $f(t, u)$ using its interpolation polynomial (backward Newton interpolation) span over previous values:

$$f(t_i, u_i), f(t_{i-1}, u_{i-1}), \dots$$



- **example for two-step method**: we consider only the last two points f_i and f_{i-1}

- interpolation polynomial is 1-st degree

$$f(t, u) = \frac{t - t_{i-1}}{\Delta t} f_i + \frac{t_i - t}{\Delta t} f_{i-1} + O(\Delta t^2)$$

- this formula interpolates the integrand while integration gives

$$u_{i+1} = u_i + \Delta t \left(\frac{3}{2} f_i - \frac{1}{2} f_{i-1} \right) + O(\Delta t^3)$$

- the general formula for k-step ABk method is written as follows

$$u_{i+1} = u_i + \sum_{r=1}^k \beta_r f_{i-r+1} + O(C \cdot \Delta t^{k+1})$$

- the k-step AB method has k-order of accuracy (global error)
- this method is explicit and requires only one calculation of $f(t, u)$ per time step while RK counterpart RK2 needs two calls to $f(t, u)$ - this is our gain in computational efficiency
- Adams-Bashforth methods has larger constant standing at leading truncation term than these in RK methods

Adams-Moulton method (implicit)

Derivation of difference formula proceeds the same way as for AB method, but additional point f_{i+1} is attached to the interpolating polynomial – it is an implicit method. General formula takes the following form

$$u_{i+1} = u_i + \beta_0 f_{i+1} + \sum_{r=1}^k \beta_r f_{i-r+1} + O(C \cdot \Delta t^{k+2})$$

- due to attaching one more (unknown) point to interpolation it becomes $k+1$ order method
- by definition AM method is implicit, however it is used as an explicit one by combining it with one of AB methods in the **predictor-corrector** scheme
- parameters of the lowest order AB and AM methods

AB : $p = k$ (order)

k	β_1	β_2	β_3	β_4	C
1					1
2	$\frac{3}{2}$	$-\frac{1}{2}$			$\frac{5}{12}$
3	$\frac{23}{12}$	$-\frac{4}{3}$	$\frac{5}{12}$		$-\frac{3}{8}$
4	$\frac{55}{24}$	$-\frac{59}{24}$	$\frac{37}{24}$	$-\frac{3}{8}$	$\frac{251}{720}$

AM : $p = k + 1$ (order)

k	β_0	β_1	β_2	β_3	C
0	1				$\frac{1}{2}$
1	$\frac{1}{2}$	$\frac{1}{2}$			$-\frac{1}{12}$
2	$\frac{5}{12}$	$\frac{2}{3}$	$-\frac{1}{12}$		$\frac{1}{24}$
3	$\frac{3}{8}$	$\frac{19}{24}$	$-\frac{5}{24}$	$\frac{1}{24}$	$-\frac{19}{720}$

- C constant in AB methods are greater than those in RK methods
- but for AM methods these are 1 order of magnitude smaller (better accuracy)

Predictor-Corrector method

- the use of AM methods as an implicit one is rather pointless, because their regions of stability are limited on the left complex half-plane – they are not the A-stable methods (as e.g. trapezoidal or Backward Euler)
- many computer experiments show however that if the value u_{i+1} calculated with AB method is plugged as an approximated value to AM formula then one gets better (corrected) result (recall that in the same way we derived the middle point formula for RK2 method)

Predictor stage (AB):

$$u_{i+1}^P = u_i + \Delta t \sum_{r=1}^{k_P} \beta_r^P f(t_{i-r+1}, u_{i-r+1})$$

Corrector stage (AM):

$$u_{i+1}^C = u_i + \Delta t \beta_0^C f(t_{i+1}, u_{i+1}^P) + \Delta t \sum_{r=1}^{k_C} \beta_r^C f(t_{i-r+1}, u_{i-r+1})$$

$$u_{i+1} = u_{i+1}^C \quad \bullet \text{ it gives smaller error than only AB call}$$

This process can be generally formulated as

$$\text{Predict} \rightarrow \underbrace{\text{Evaluate} \rightarrow \text{Correct}}_{\text{repeat m times}} \rightarrow \text{Evaluate} \implies P(EC)^m E$$

and analogously to the Picard iteration this scheme might be further generalized as iteration to fixed point

$$\lim_{m \rightarrow \infty} P(EC)^m E = u_{i+1} \quad (\text{better accuracy})$$

However, usually the parameter m is limited to 1, since other values need remarkably larger computational effort.

Remark: Predictor-Corrector for $m=1$ calls the r.h.s. of ODE twice, irrespective of its order, while RK4 needs 4 calls

Controlling local error in Predictor-Corrector scheme (Milne device)

If Predictor-Corrector is used with AB and AM methods of the same order, that is k and $k-1$ steps, then both results approximately differ in their truncation errors [below the present solution u_i is considered to be exact]

$$\begin{aligned}
 (1) \quad u_{i+1}^P &= \underbrace{u(t_{i+1})}_{\text{exact}} - \underbrace{\Delta t^{k+1} C_k^P u^{(k+1)}(t_i)}_{\text{leading truncation term}} + O(\Delta t^{k+2}) \quad / \cdot C_{k-1}^C \\
 (2) \quad u_{i+1}^C &= \underbrace{u(t_{i+1})}_{\text{exact}} - \underbrace{\Delta t^{k+1} C_{k-1}^C u^{(k+1)}(t_i)}_{\text{leading truncation term}} + O(\Delta t^{k+2}) \quad / \cdot C_k^P
 \end{aligned}$$

multiplication of (1) and (2) followed by their subtraction gives estimation of **local truncation error**

$$\varepsilon_{loc} = u(t_{i+1}) - u_{i+1} = \frac{C_{k-1}^C}{C_{k-1}^C - C_k^P} (u_{i+1}^P - u_{i+1}^C) + O(\Delta t^{k+2})$$

Such monitoring of the truncation error is conventionally used for the step size control in stiff problems. The change of step size in multi-step methods is not as easy as for the one-step methods.

Initial condition for multi-step method

Starting the one-step method is easy, one needs to provide the single value (solution) at starting point

$$ODE1 : \quad \frac{du}{dt} = f(t, u), \quad u_0 = u(t_0)$$

but for multistep method we need the k values at time steps: $u_0, u_1, u_2, \dots, u_{k-1}$ (and usually we know only one: u_0)

The missing values can be found in two ways:

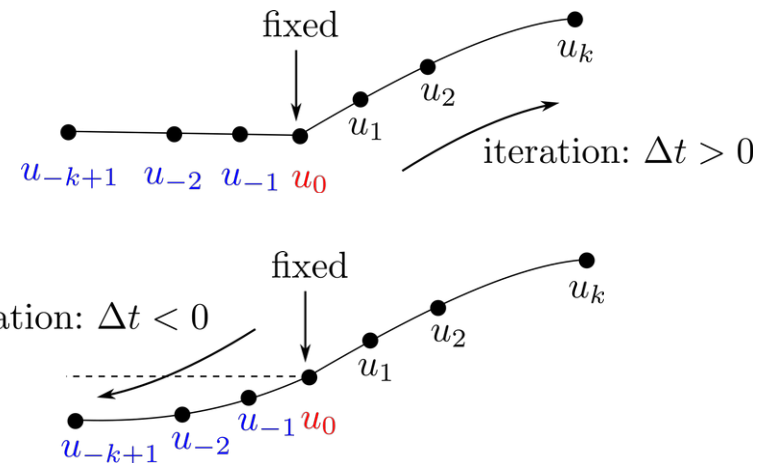
- just use the Runge-Kutta k-order method to generate the subsequent k-1 solutions, RK method must have the same order as predictor-corrector (!!!) so as to avoid losing accuracy
- applying predictor-corrector method with alternately positive and negative values of time step, this is done recursively as follows

(1) set the unknown preceding values to u_0 $u_{-1} = u_{-2} = \dots = u_{-k+1} = u_0$

(2) set $\Delta t > 0$ and estimate subsequent needed values for initial condition

(3) set $\Delta t < 0$ and improve the preceding values

(4) repeat steps (2) and (3) until initial values $u_1, u_2, \dots, u_{k-1}$ converge (few to several iterations are needed)



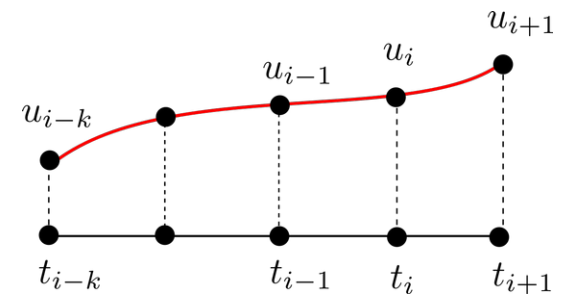
Backward Differentiation Formulae (BDF) - implicit

For a reference point u_{i+1} one needs to determine the number of steps taken back in derived formula: $k+1$, then the interpolating Lagrange polynomial $W(t)$ is constructed upon all involved pairs (t, u)
 - the number of points defines the degree of polynomial and order of BDF

$$(t_{i-k}, u_{i-k}), \dots, (t_{i-1}, u_{i-1}), (t_i, u_i), (t_{i+1}, u_{i+1})$$

$$u(t) = W(t) + O(\Delta t^{k+2})$$

$$W(t) = \sum_{r=-k}^1 \prod_{\substack{j=-k \\ j \neq r}}^1 \frac{(t - t_{i+j})}{(t_{i+r} - t_{i+j})} u_{i+r}$$



Because $W(t)$ interpolates the solution $u(t)$ its time derivative would approximate the r.h.s. of ODE1 at time step t_{i+1}

$$\frac{du}{dt} = f(t, u) \quad \Rightarrow \quad \left. \frac{dW}{dt} \right|_{t=t_{i+1}} = f(t_{i+1}, u_{i+1})$$

example: 2-step (3-point) formula

$$W(t) = \frac{(t - t_i)(t - t_{i+1})}{(t_{i-1} - t_i)(t_{i-1} - t_{i+1})} u_{i-1} + \frac{(t - t_{i-1})(t - t_{i+1})}{(t_i - t_{i-1})(t_i - t_{i+1})} u_i + \frac{(t - t_{i-1})(t - t_i)}{(t_{i+1} - t_{i-1})(t_{i+1} - t_i)} u_{i+1} + O((t - t_{i-1})^3)$$

$$\left. \frac{dW}{dt} \right|_{t=t_{i+1}} = \frac{1}{\Delta t} \left[\frac{3}{2} u_{i+1} - 2u_i + \frac{1}{2} u_{i-1} \right] + O(\Delta t^2) = f(t_{i+1}, u_{i+1}) \quad \leftarrow \text{BDF method of order 2}$$

In similar way other higher order method can be derived, but only these of order up to 6 are A(α)-stable

- 1-order method $u_{i+1} = u_i + \Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^2)$ (Backward Euler)
- 2-order method $u_{i+1} = \frac{4}{3}u_i - \frac{1}{3}u_{i-1} + \frac{2}{3}\Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^3)$
- 3-order method $u_{i+1} = \frac{18}{11}u_i - \frac{9}{11}u_{i-1} + \frac{2}{11}u_{i-2} + \frac{6}{11}\Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^4)$
- 4-order method $u_{i+1} = \frac{48}{25}u_i - \frac{36}{25}u_{i-1} + \frac{16}{25}u_{i-2} - \frac{3}{25}u_{i-3} + \frac{12}{25}\Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^5)$
- 5-order method $u_{i+1} = \frac{300}{137}u_i - \frac{300}{137}u_{i-1} + \frac{200}{137}u_{i-2} - \frac{75}{137}u_{i-3} + \frac{12}{137}u_{i-4} + \frac{60}{137}\Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^6)$
- 6-order method $u_{i+1} = \frac{120}{49}u_i - \frac{150}{49}u_{i-1} + \frac{400}{147}u_{i-2} - \frac{75}{49}u_{i-3} + \frac{24}{49}u_{i-4} - \frac{10}{147}u_{i-5} + \frac{20}{49}\Delta t f(t_{i+1}, u_{i+1}) + O(\Delta t^7)$

Remark: The use of any of these implicit formulae requires iterative finding solution of nonlinear equation – Picard/Newton/Secant subiteration

Time step control

When ODE is solved with one of difference schemes the first question which one poses is :
what should be an optimal time step?

This can be adjusted empirically by carrying out several test simulations i.e. the process is supervised by the user being hence very involving. Although some decision is made, the chosen time step might not be optimal, let's see why by considering simple Forward Euler method

$$u_{i+1} = u_i + \Delta t f(t_i, u_i) + \underbrace{\frac{\Delta t^2}{2} \left. \frac{d^2 u}{dt^2} \right|_{t \in [t_i, t_{i+1}]} }_{\text{truncation error}}$$

The step size is fixed for some chosen time interval $[t_i, t_{i+1}]$, however in another interval the second derivative may change value remarkably – growing or shrinking.

In order to prevent potential numerical instabilities the time step is chosen small being far from the optimal one.

To get rid off this problem an **adaptive time step control** methods may be used. These can be easily implemented in one-step methods while the multistep methods require more extended algorithms

Any method implementing this functionality needs information about

- estimation of local truncation error
- the order (p) of integration method

Local truncation error can be approximated with two methods

- **step-dubbling**
- **embedding** (e.g. Runge-Kutta-Fehlberg method, aka RKF45)

Adaptive time step control

Each difference scheme has its local truncation error with a leading term in the following form

$$\varepsilon = C \cdot \Delta t^{p+1} \quad (\text{p - order of method})$$

In adaptive methods, this error can not exceed some fixed upper limit called tolerance (TOL).

With fixed TOL factor one can estimate an optimal time step Δt_{new}

$$TOL = C \cdot \Delta t_{\text{new}}^{p+1}$$

By dividing both expressions and adding heuristic safety factor S we get the relation between both time steps

$$\Delta t_{\text{new}} = S \cdot \Delta t \left(\frac{TOL}{\varepsilon} \right)^{\frac{1}{p+1}} \quad (S = 0.9)$$

the factor S shall prevent us, to some extent, from balancing on the border of stability region in next time step by localizing the solution within stability region.

Additionally the following rules must be added to the adaptive algorithm

$$\frac{TOL}{\varepsilon} > 1? \begin{cases} \text{YES} & \implies \text{set } \Delta t \leftarrow \Delta t_{\text{new}} \text{ and use in next step} \\ \text{NO} & \implies \text{set } \Delta t \leftarrow \Delta t_{\text{new}} \text{ and repeat previous step} \end{cases}$$

Step doubling

The idea is following: we use the method of order p to find two solutions

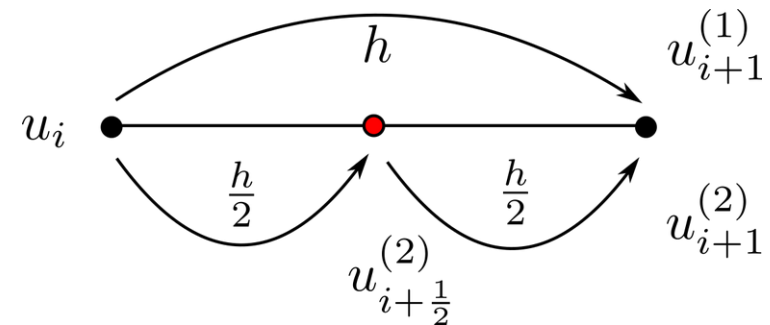
- first by making one long step Δt
- second (more accurate) by making two smaller steps $\Delta t/2$

from comparison of both we infer an approximation of local error and we improve the solution.

$$(1) \quad u_{i+1}^{(1)} = \tilde{u}(t_{i+1}) + C \cdot \Delta t^{p+1}$$

$$(2) \quad u_{i+1}^{(2)} = \tilde{u}(t_{i+1}) + 2 \cdot C \left(\frac{\Delta t}{2} \right)^{p+1}$$

$\tilde{u}(t_{i+1})$ unknown exact solution for initial u_i



- subtraction (2)-(1) eliminates exact solution and allows to determine the error factor C

$$C = \frac{u_{i+1}^{(2)} - u_{i+1}^{(1)}}{\Delta t^{p+1}(2^{-p} - 1)}$$

- plugging C into (2) gives estimation of local error

$$\varepsilon \approx \tilde{u}(t_{i+1}) - u_{i+1}^{(2)} = \frac{u_{i+1}^{(2)} - u_{i+1}^{(1)}}{2^p - 1}$$

- local error can be used to improve the second (better) solution

$$u_{i+1} = u_{i+1}^{(2)} + \frac{u_{i+1}^{(2)} - u_{i+1}^{(1)}}{2^p - 1}$$

- pseudocode for adaptive time step control method using step-doubling

```

initialization:  $t_{max}$ ,  $n$ ,  $u = u(t = 0)$ ,  $p - order$ ,
                 $S = 0.9$ ,  $TOL = 10^{-6}$ 
                 $\Delta t = \frac{t_{max}}{n}$ 
                 $t = 0$ 
WHILE  $t < t_{max}$  DO

    adjust the time step:
    DO
         $u^{(1)} \leftarrow procedure(\Delta t, u)$ 
         $u_{\frac{1}{2}}^{(2)} \leftarrow procedure(\Delta t/2, u)$ 
         $u^{(2)} \leftarrow procedure(\Delta t/2, u_{\frac{1}{2}}^{(2)})$ 
         $\varepsilon \leftarrow \frac{u^{(2)} - u^{(1)}}{2^p - 1}$ 
         $\Delta t_{new} \leftarrow S \cdot \Delta t \cdot \left( \frac{TOL}{|\varepsilon|} \right)^{\frac{1}{p+1}}$ 
         $\Delta t_{old} \leftarrow \Delta t$ 
         $\Delta t \leftarrow \Delta t_{new}$ 
    WHILE  $\frac{TOL}{|\varepsilon|} < 1$ 

     $u \leftarrow u^{(2)} + \varepsilon$ 
     $t = t + \Delta t_{old}$ 

    send output:  $t$ ,  $u$ 
END DO

```

Multiscale problems – stiffness in ODE

The issue of multiscaling occur in many areas of physics and concerns the linear and nonlinear problems. Because the analysis for the linear system ODE is simpler we start considerations on stiffness for this type ODE.

The linear **system of ordinary differential equations** (SODE) in vector representation

$$\frac{d\vec{u}}{dt} = \mathbf{A}\vec{u}, \quad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad \vec{u} \in \mathbb{R}^n$$

Assuming the matrix of the system is diagonalizable, the solution of SODE can be expanded in this vector basis

$$\mathbf{A}\vec{v}_j = \lambda_j \vec{v}_j \quad (\vec{w}_j^T \mathbf{A} = \lambda_j \vec{w}_j^T) \quad \Rightarrow \quad \vec{u}(t=0) = \sum_{j=1}^n \alpha_j(0) \vec{v}_j, \quad \alpha_j(0) = \vec{w}_j^T \vec{u}_0$$

$$\vec{u}(t > 0) = \sum_{j=1}^n \alpha_j(t) \vec{v}_j,$$

The time-dependence of coefficients α_j we find by plugging expansion $u(t)$ into SODE and isolating particular coefficient

$$\frac{d\vec{u}}{dt} = \sum_{j=1}^n \frac{d\alpha_j}{dt} \vec{v}_j \equiv \mathbf{A}\vec{u} = \sum_{j=1}^n \alpha_j \lambda_j \vec{v}_j$$

$$\sum_{j=1}^n \underbrace{\left(\frac{d\alpha_j}{dt} - \alpha_j \lambda_j \right)}_{=0} \vec{v}_j = \vec{0}$$

$$\alpha_j(t) = \alpha_j(0) e^{\lambda_j t}$$

$$\frac{d\vec{u}}{dt} = \mathbf{A}\vec{u}$$

$$\vec{u}(t) = \sum_{j=1}^n \alpha_j(0) e^{\lambda_j t} \vec{v}_j$$

Now let's modify SODE by adding the terms depending solely on time

$$\frac{d\vec{u}}{dt} = \mathbf{A}\vec{u} + \vec{\varphi}(t), \quad \mathbf{A} \in \mathbb{R}^{n \times n}, \quad \vec{u}, \vec{\varphi} \in \mathbb{R}^n$$

For $\Phi(t)$ being the particular solution of SODE, the general solution can be written as

$$\vec{u}(t) = \sum_{j=1}^n \alpha_j(0) e^{\lambda_j t} \vec{v}_j + \vec{\Phi}(t)$$

this solution can be considered as a combination of transient part (sum) decaying with time and the stationary part (Φ) to which the solution tends.

Very informal definition of stiffness of ODE (strict definition doesn't exist)

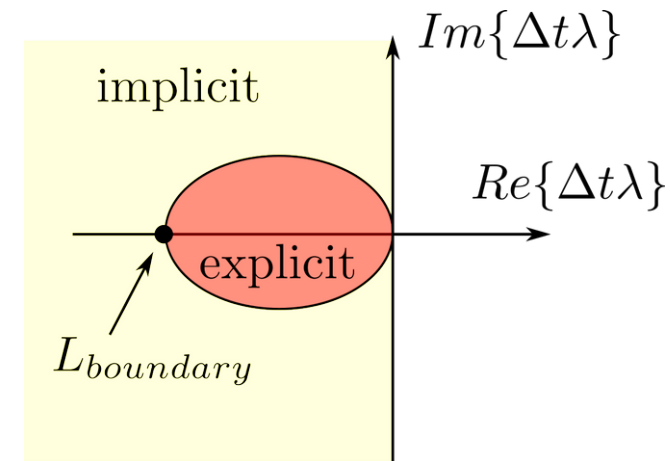
For a set of (negative) eigenvalues of A

$$\lambda_j < 0, \quad |\lambda_1| < |\lambda_2| < \dots < |\lambda_{max}|$$

any numerical scheme with bounded stability region used to solve linear SODE requires time step subjected to constraint imposed by largest magnitude eigenvalue λ_{max} .

$$L_{boundary} \leq \operatorname{Re}\{\Delta t \cdot \lambda_{max}\} < 0$$

$$0 < \Delta t \leq \frac{L_{boundary}}{\lambda_{max}}$$



Remarks:

- **stiffness paradox** – step size must be in corcondance with fast decaying component in ODE which is almost flat over a large range of time but still influences the stability
- because explicit methods largely fail in solving the stiff problems, the implicit ones are recommended instead

example: SODE

$$\frac{d^2u}{dt^2} + (\lambda_0 + \lambda_1) \frac{du}{dt} + \lambda_0 \lambda_1 u = 0, \quad \underbrace{u(0) = 2, \quad \frac{du}{dt}|_{t=0} = -\lambda_0 - \lambda_1}_{\text{initial condition}}$$

$$\text{set : } u^0 = u$$

$$\begin{cases} \frac{du^0}{dt} = u^1 \\ \frac{du^1}{dt} = -\lambda_0 \lambda_1 u^0 - (\lambda_0 + \lambda_1) u^1 \end{cases} \implies \frac{d\vec{u}}{dt} = \mathbf{A} \vec{u} = \begin{bmatrix} 0 & 1 \\ -\lambda_0 \lambda_1 & -(\lambda_0 + \lambda_1) \end{bmatrix} \vec{u}$$

$$\text{Eigenvalues of A: } -\lambda_0, -\lambda_1$$

$$\text{stiffness if: } \lambda_1 \gg \lambda_0$$

$$\text{exact solution: } u(t) = \exp(-\lambda_0 \cdot t) + \exp(-\lambda_1 \cdot t)$$

$$\text{parameters used in calculations: } \lambda_0 = 1, \quad \lambda_1 = 20$$

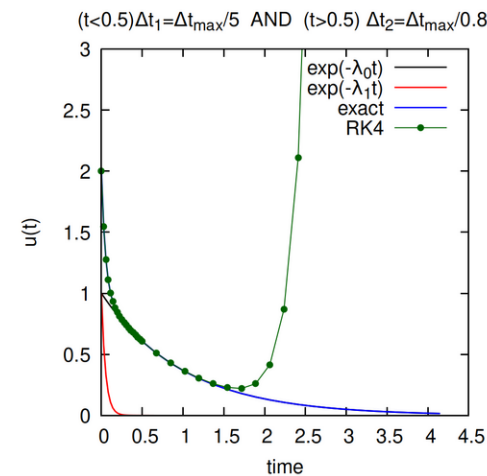
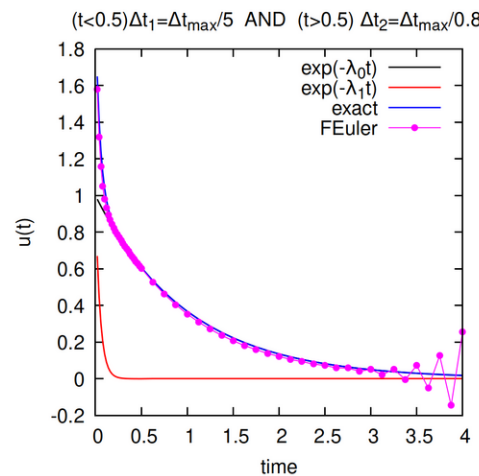
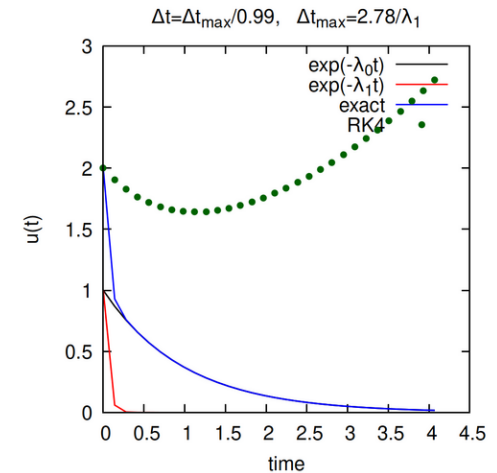
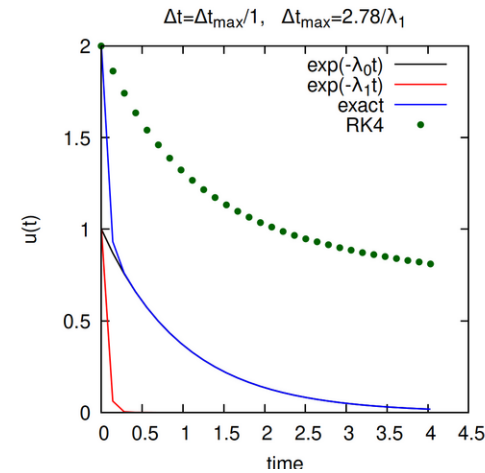
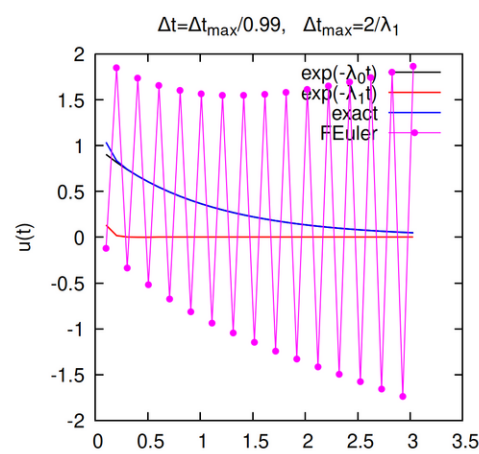
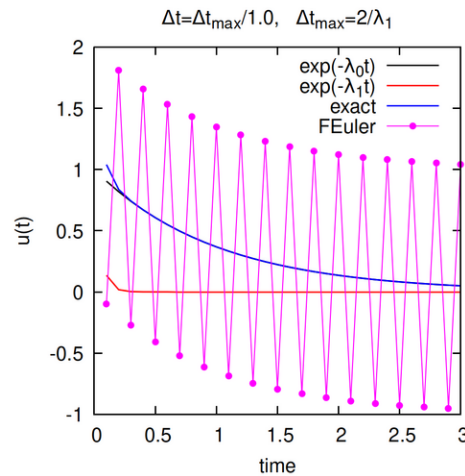
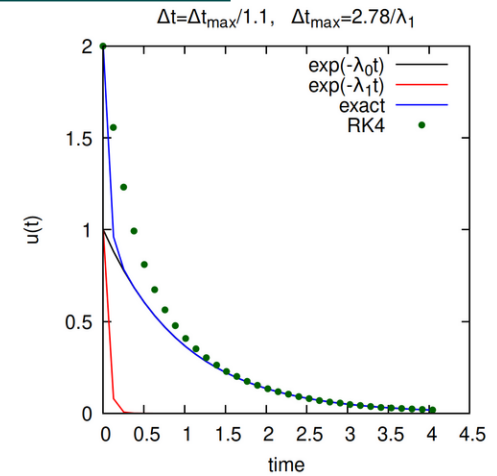
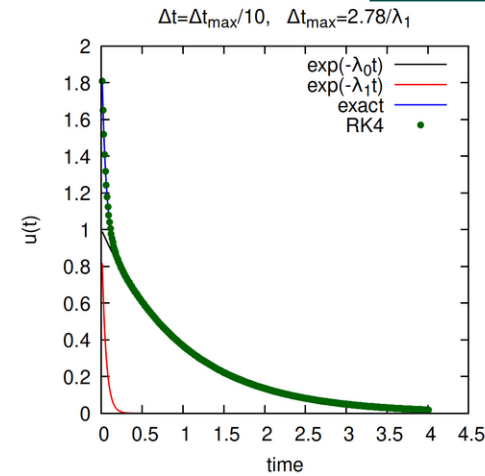
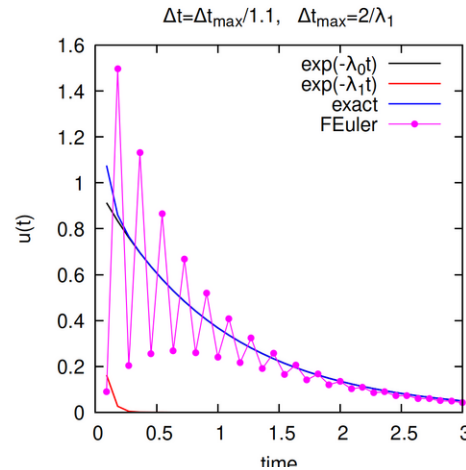
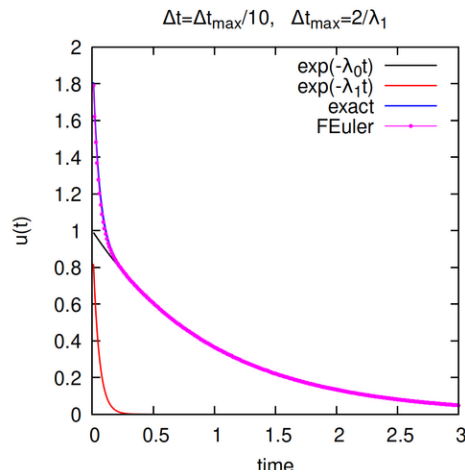
SODE with these parameters becomes stiff

Let's see what happens if Forward Euler and RK4 are used with fixed time step size →

constant Δt :

FORWARD EULER

RK4

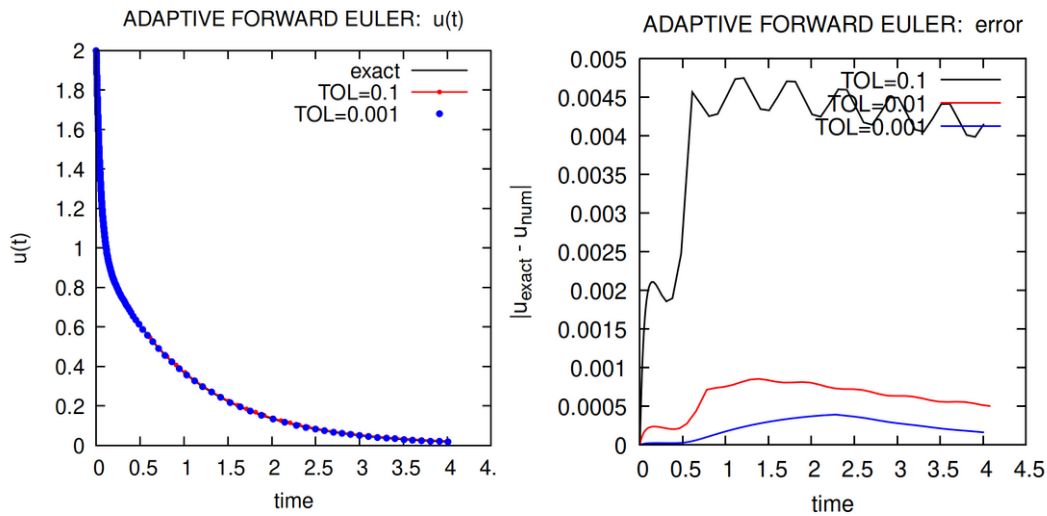


$$\lambda_1 = 20$$

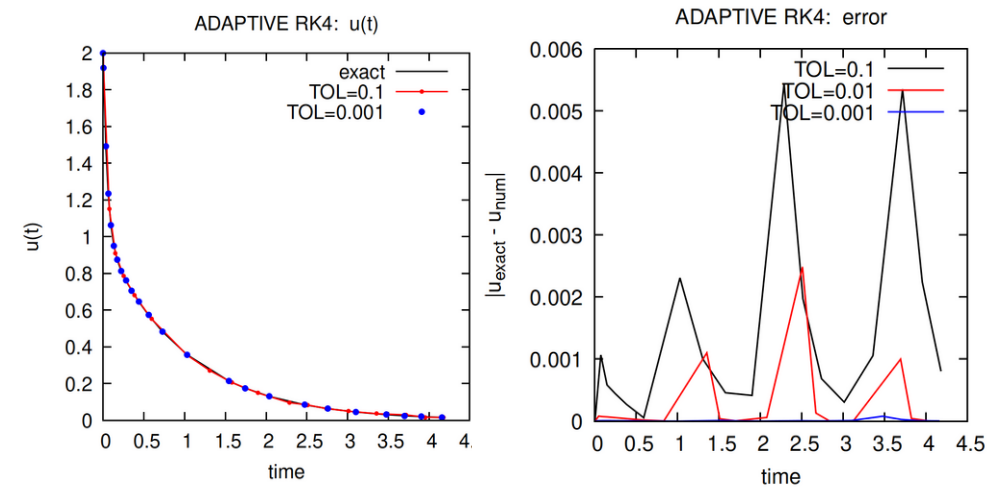
$$e^{-20 \cdot 0.5} = 4.5 \cdot 10^{-5}$$

- even though second component doesn't contribute to solution for $t > 1/2$ it still limits the size of time step

ADAPTIVE FORWARD EULER

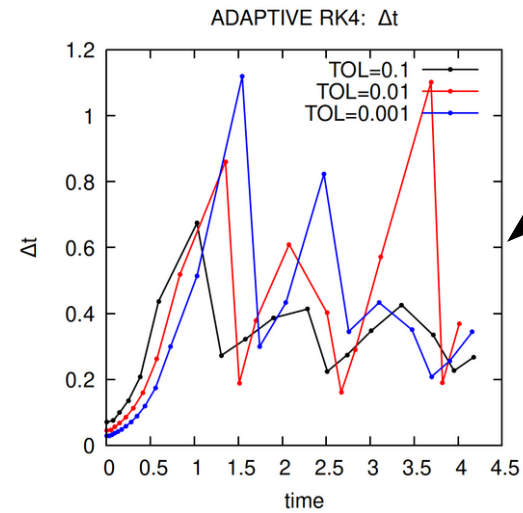
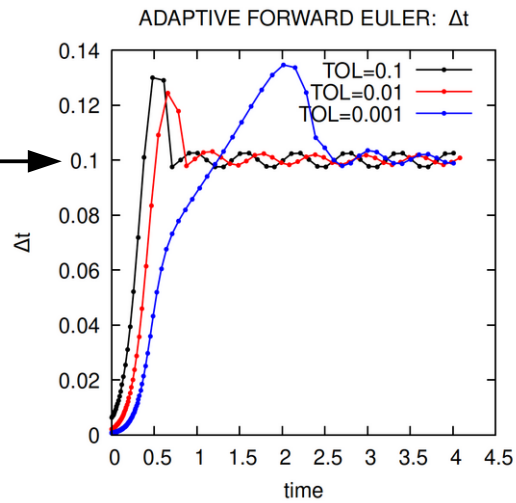


ADAPTIVE RK4



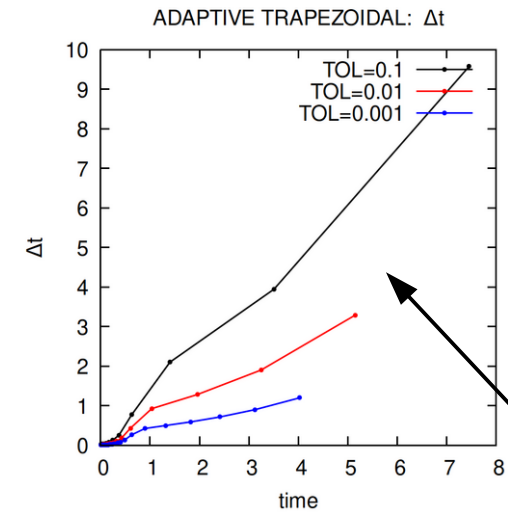
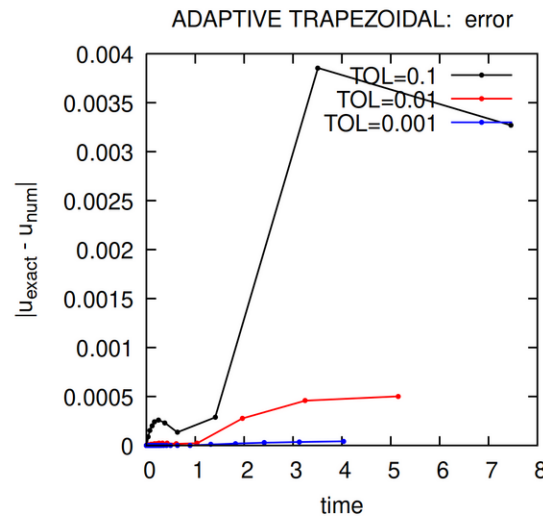
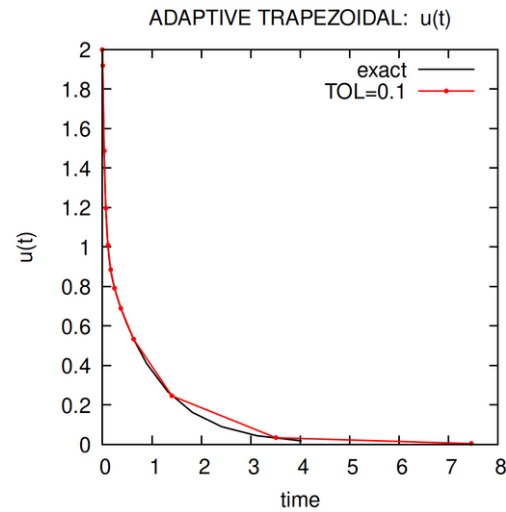
FE tests the stability condition

$$\Delta t \approx \frac{2}{\lambda_1} = 0.1$$



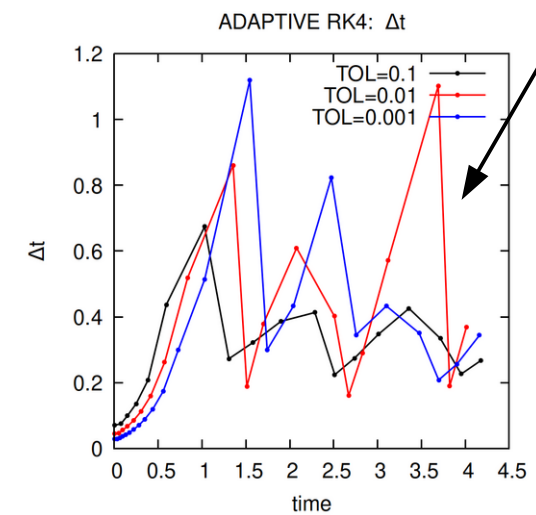
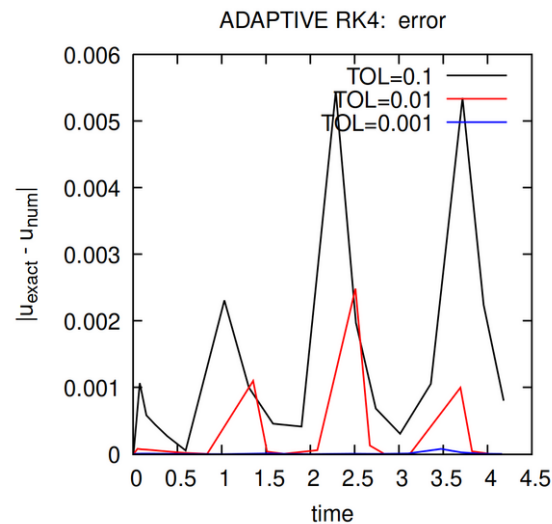
RK4 has higher accuracy and set the Δt a few times larger than F-Euler

ADAPTIVE TRAPEZOIDAL + NEWTON ITERATION



- Implicit trapezoidal method sets larger time step size than explicit RK4

ADAPTIVE RK4



2-nd order ODE: Verlet methods

Original version of method was derived in 1967 by Loup Verlet for use in molecular dynamics simulations. Such simulations need very long time calculations of trajectories, velocities and energies for multitude of particles (**thousands to millions**) and interactions, this definitely requires high accuracy numerical integrators.

For Newton dynamics we get 2-nd order ODE $F = ma = m \frac{d^2 r}{dt^2} \implies r(t) = ?$

The formal derivation of integral scheme is directly given for the position vector r . From forward and backward Taylor expansion we get

$$(1) \quad r(t + \Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}\frac{d^3 r}{dt^3} + O(\Delta t^4)$$

$$(2) \quad r(t - \Delta t) = r(t) - v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 - \frac{1}{6}\frac{d^3 r}{dt^3} + O(\Delta t^4)$$

theoretically having trajectory $r(t)$ at hand we know everything about the system's dynamics

Addition of (1) and (2) gives **standard Verlet's formula**

$$r(t + \Delta t) = 2r(t) - r(t - \Delta t) + a(t)\Delta t^2 + O(\Delta t^4)$$

high accuracy despite very simple scheme

Remarks:

- (**pros**) high order
- (**pros**) time reversible $\Delta t \rightarrow -\Delta t$
- (**pros**) symplectic (conserves energy for Hamiltonian dynamics)
- (**cons**) this difference scheme is not self-starting – first step must be done with another method
- (**cons**) we need velocity for kinetic energy, this must be calculated independently (with lower accuracy)

Velocity can be calculated by applying the finite difference

$$V(t) = \frac{x(t + \Delta t) - x(t - \Delta t)}{2\Delta t} + O(\Delta t^2)$$

two orders lower accuracy than for position

Velocity Verlet method

The original Verlet method is not self-starting and doesn't estimate well the velocity.
To get rid of these drawbacks we need to manipulate a little the expression for position vector.

Let's write the backward Taylor expansion for present position $r(t)$

$$(1) \quad r(t) = r(t + \Delta t) - v(t + \Delta t)\Delta t + \frac{1}{2}a(t + \Delta t)\Delta t^2 + O(\Delta t^3)$$

and combine it with another forward expansion

$$(2) \quad r(t + \Delta t) = r(t) + \underbrace{v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2}_{\Delta t[V(t) + a(t)\Delta t/2]} + O(\Delta t^3)$$

this gives the velocity in intermediate position

$$(3) \quad V(t + \Delta t) = V(t) + \frac{1}{2}[a(t) + a(t + \Delta t)]\Delta t + O(\Delta t^3)$$

$a(t + \Delta t)$ - needs to be estimated but it's very simple if we have $x(t + \Delta t)$
– use the expression for force

$$a(t + \Delta t) = \frac{F(t + \Delta t, r(t + \Delta t))}{m}$$

Combination of (3) and (2) leads straightforwardly to following scheme

$$\text{velocity Verlet scheme} \left\{ \begin{array}{l} V\left(t + \frac{\Delta t}{2}\right) = V(t) + \frac{\Delta t}{2}a(t) \\ r(t + \Delta t) = r(t) + V\left(t + \frac{\Delta t}{2}\right)\Delta t + O(\Delta t^3) \\ V(t + \Delta t) = V\left(t + \frac{\Delta t}{2}\right) + \frac{1}{2}a(t + \Delta t)\Delta t + O(\Delta t^3) \end{array} \right.$$

Remarks:

- high accuracy for both, position and velocity
- self-starting (one-step method)

Position Verlet method

In a similar way as for velocity-Verlet we can derive second type of Verlet method starting from expansion for velocity, then we get the following scheme

position Verlet scheme

- high accuracy
- self-starting

$$\left\{ \begin{array}{l} r\left(t + \frac{\Delta t}{2}\right) = r(t) + V(t)\frac{\Delta t}{2} + O(\Delta t^2) \\ v(t + \Delta t) = v(t) + a\left(t + \frac{\Delta t}{2}\right)\Delta t + O(\Delta t^3) \\ r(t + \Delta t) = r\left(t + \frac{\Delta t}{2}\right) + V(t + \Delta t)\frac{\Delta t}{2} + O(\Delta t^3) \end{array} \right.$$

Final remarks:

- Verlet's methods could be used when the time dynamics of the system is defined with second order time derivative (usually for Newtonian forces but also in solving the wave equations $\rightarrow$ PDE)
- the **standard** and **velocity Verlet methods** give exactly the same trajectories $r(t)$, provided that the first step for the former is prepared as

$$r(\Delta t) = r(0) + \Delta t v(0) + \frac{\Delta t^2}{2} a(0) + O(\Delta t^3)$$

- **time reversibility** can be used in estimation of numerical errors, e.g. due to finite size of time step, then simulation goes in time for some $t_{\max}$, then substitution $\Delta t \rightarrow -\Delta t$ is made and simulation is repeated in reverse direction, the difference in trajectories at $t=0$ are a measure of errors