The Halfway Way:
A fresh look at the midpoint method

Catalin Trenchea, Wenlong Pei, John Burkardt
ICAM Conference on Applied and Computational Mathematics
Honoring Terry Herdman on his retirement
01-03 June 2022
Virginia Tech

The Executive Summary

- The midpoint method is second order and absolutely stable;
- It is B-stable;
- It preserves linear and quadratic conservation quantities;
- It produces reliable error estimates;
- Safe time-steps are calculated accurately, efficiently, and adaptively;
- Existing backward Euler codes upgrade with one line of new code;
- C, C++, Fortran, FreeFem, MATLAB, Octave, Python, R versions.

Multiple Identities

Methods are for ODE's, rules for numerical quadrature.

	f(mid)	$\frac{1}{2}$(f(left)+f(right))
explicit:	explicit midpoint method	Heun's method or Improved Euler method
implicit:	implicit Runge-Kutta 2 or **implicit midpoint method**	trapezoidal method

From now on, "midpoint method" refers to *implicit midpoint method.*

Two Ways to Look at It

The implicit midpoint method can be seen as:

$$y_{n+1} = y_n + \tau_n f(t_{n+1/2}, y_{n+1/2})$$

or as Backward and Forward Euler steps of size $\frac{\tau_n}{2}$:

$$y_{n+1/2} = y_n + \frac{\tau_n}{2} f(t_{n+1/2}, y_{n+1/2}) \quad \text{(BE: Backward Euler)}$$

$$y_{n+1} = y_{n+1/2} + \frac{\tau_n}{2} f(t_{n+1/2}, y_{n+1/2}) \quad \text{(FE: Forward Euler)}$$

The second step can be rewritten simply as:

$$y_{n+1} = 2y_{n+1/2} - y_n \quad \text{(FE: Forward Euler)}$$

so the implicit problem only needs to be solved once, in BE.

The resulting method can be designated as (BEFE):

$$y_{n+1/2} = y_n + \frac{\tau_n}{2} f(t_{n+1/2}, y_{n+1/2}) \quad \text{(BE: Backward Euler)}$$

$$y_{n+1} = 2y_{n+1/2} - y_n \quad \text{(FE: Forward Euler)}$$

and admits a generalization to Cauchy's one-leg θ method:

$$y_{n+\theta_n} = y_n + \theta_n \tau_n f(t_{n+\theta_n}, y_{n+\theta_n})$$

$$y_{n+1} = \frac{1}{\theta_n} y_{n+\theta_n} - (\frac{1}{\theta_n} - 1) y_n$$

Stability

The θ-method for $\frac{1}{2} \leq \theta_n \leq 1$, and the BEFE special case are unconditionally stable, A-stable, and B-stable.

We say a method is **B-stable** if, for all u, v elements of a Banach or Hilbert space, and $\forall f()$ for which $\langle f(u) - f(v), u - v \rangle \leq 0$, we have $||y_{n+1} - z_{n+1}|| \leq ||y_n - z_n||$, for any two sequences y and z of approximations computed with the method, and any index n.

B-stability implies A-stability.

Error Estimates for Adaptive Stepsize

For a smooth exact solution $y(x)$, the local truncation error for BEFE is

$$T_{n+1} \equiv y(t_{n+1}) - y_{n+1} = \frac{1}{24}\,\tau_n^3\, y'''(t_n + 1/2) + \mathcal{O}(\tau_n^5)$$

For a given local error tolerance `tol`, propose the next time step as

$$\tau_{n+1} = \kappa\,\tau_n \Big(\frac{\text{tol}}{||\,T_{n+1}||}\Big)^{\frac{1}{3}}$$

where the safety factor $\kappa \leq 1$.

How to go MAD: Midpoint Adaptive method:

t_0, y_0, $\texttt{tol}$, T, κ given.

t_1, y_1, τ_0 from one step second-order method in convergence range

$t^{\text{new}} = t_1$, $\tau^{\text{new}} = \tau_0$, $n = 1$

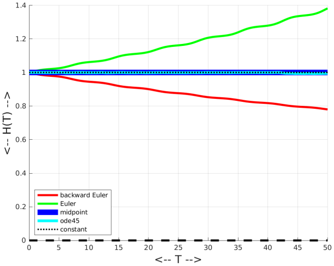
while $t_n \leq T$ **do**

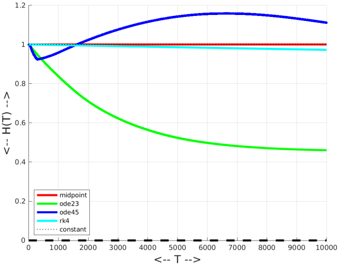
- $\tau_n \leftarrow \tau^{\text{new}}$;
- evaluate y_{n+1} with the midpoint rule;
- evaluate $\widehat{T}_{n+1}$;
- $\tau^{\text{new}} \leftarrow \kappa\, \tau_n \left|\texttt{tol}/\|\widehat{T}_{n+1}\|\right|^{\frac{1}{3}}$;
- **if** $\|\widehat{T}_{n+1}\| \leq \texttt{tol}$ **then**
  - $t_{n+1} \leftarrow t_n + \tau^{\text{new}}$,
  - $n \leftarrow n + 1$
- **end**

end

Test: Rigid Body Rotation

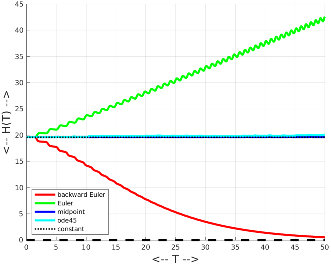
Conservation: $H(t) = u^2 + v^2 + w^2$

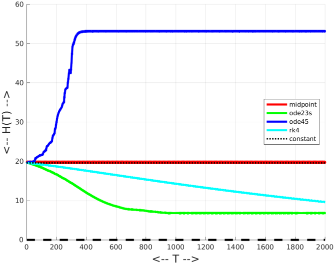




Test: Nonlinear Pendulum

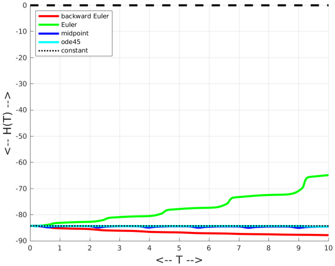
Conservation: $H(t) = \frac{mg}{l}(1-\cos(u)) + \frac{1}{2}mv^2$

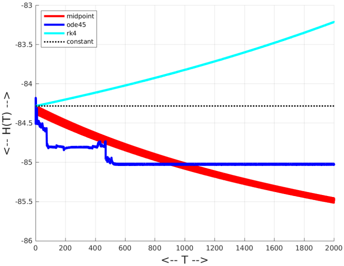




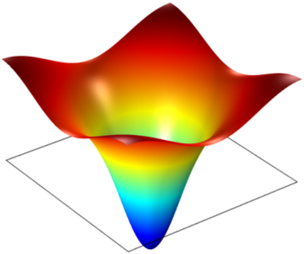
Test: Predator Prey

Conservation: $H(t) = \delta u - \gamma \log(u) + \beta v - \alpha \log(v)$
(not quadratic!)





Implicit Solvers Must Handle Nonlinear Equations



Any implicit ODE method must reliably solve a sequence of systems of nonlinear equations. Given the small stepsizes of a typical ODE method, the previous ODE solution is often a good first approximation to the solution at a new (but very close) time.

A simple method that usually works is to apply a fixed point iteration.

The developers of MINPACK provided the function `hybrd()` for solving general systems of nonlinear equations, and versions of this code are available in MATLAB, Python, and R under the name `fsolve()`.

Fixed Point Iteration

Ada Lovelace: *"The calculation will eat its own tail."*

```
tm = t0 + 0.5 * dt
ym = y0 + 0.5 * dt * f(t0,y0)

for j = 1 : it_max
  ym = y0 + 0.5 * dt * f(tm,ym)
end

t1 = t0 + dt
y1 = 2 * ym - y0
```

Using fsolve()

If you're close enough, you can't miss!

```
th = to + 0.5 * dt;
yh = yo + 0.5 * dt * ( f ( to, yo ) )';
yh = fsolve ( @(yh)residual(f,to,yo,th,yh), yh );
     ...
function value = residual ( f, to, yo, th, yh )
  value = yh - yo - ( th - to ) * ( f ( th, yh ) )';
  return
end
```

Codes available

Enter your ODE, and crank out the result!

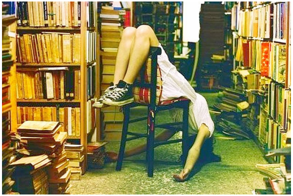
Language	fixed point	fsolve	adaptive
C	midpoint_fixed.c	midpoint.c	
C++	midpoint_fixed.cpp	midpoint.cpp	
Fortran77	midpoint_fixed.f	midpoint.f	
Fortran90	midpoint_fixed.f90	midpoint.f90	
FreeFem		midpoint.edp	
MATLAB	midpoint_fixed.m	midpoint.m	midpoint_adaptive.m
Octave	midpoint_fixed.m	midpoint.m	midpoint_adaptive.m
Python	midpoint_fixed.py	midpoint.py	midpoint_adaptive.py
R	midpoint_fixed.R	midpoint.R	

Professional Codes available

Or you may prefer your ODE to be handled by a professional!

Language	library	code
C	Gnu Scientific Library	gsl_odeiv2_step_rk2imp()
C++	Gnu Scientific Library	gsl_odeiv2_step_rk2imp()
Julia	ODE	Midpoint

References



Catalin Trenchea, John Burkardt,
Refactorization of the midpoint rule,
Applied Mathematics Letters,Volume 107, September 2020,

Catalin Trenchea, John Burkardt,
Refactorization of the midpoint rule,
Technical Report TR-MATH 20-02,
https://www.mathematics.pitt.edu/sites/default/files/midpoint3_technicalreport.pdf,

John Burkardt, Wenlong Pei, Catalin Trenchea,
A stress test for the midpoint time-stepping method,
International Journal of Numerical Analysis and Modeling, Volume 19, Number 2-3, pages 299-314, 2022.

Links

Links to source code in any language:

https://people.sc.fsu.edu/∼jburkardt/...

Language	subdirectory	Sample directory
C	c_src/	midpoint.html
C++	cpp_src/	midpoint.html
Fortran77	f77_src/	midpoint.html
Fortran90	f_src/	midpoint.html
FreeFem++	freefem_src/	midpoint.html
MATLAB	m_src/	midpoint.html
Octave	octave_src/	midpoint.html
Python	py_src/	midpoint.html
R	r_src/	midpoint.html

For most languages, there are actually several implementations: fixed point/fsolve/adaptive.

The End

- The midpoint method is powerful, accurate, and stable.
- The method is A-stable, B-stable, linearly and nonlinearly stable.
- It is a symplectic method for general Hamiltonian systems.
- The correct estimator for local truncation error only involves the differentiation defect, but not the interpolation defect.
- Implementations are provided in a variety of computing languages.