Liquid argon benchmarks

Sebastian Micluța-Câmpeanu, Mikhail Vaganov

June 19, 2020

The purpose of these benchmarks is to compare several integrators for use in molecular dynamics simulation. We will use a simulation of liquid argon form the examples of NBodySimulator as test case.

```julia
using ProgressLogging
using NBodySimulator, OrdinaryDiffEq, StaticArrays
using Plots, DataFrames, StatsPlots

function setup(t)
    T = 120.0 # K
    kb = 1.38e-23 # J/K
    ϵ = T * kb # J
    σ = 3.4e-10 # m
    ρ = 1374 # kg/m^3
    m = 39.95 * 1.6747 * 1e-27 # kg
    N = 216
    L = (m*N/ρ)^(1/3)
    R = 2.25σ
    v_dev = sqrt(kb * T / m) # m/s

    _L = L / σ
    _σ = 1.0
    _ϵ = 1.0
    _m = 1.0
    _v = v_dev / sqrt(ϵ / m)
    _R = R / σ

    bodies = generate_bodies_in_cell_nodes(N, _m, _v, _L)
    lj_parameters = LennardJonesParameters(_ϵ, _σ, _R)
    pbc = CubicPeriodicBoundaryConditions(_L)
    lj_system = PotentialNBodySystem(bodies, Dict(:lennard_jones => lj_parameters));
    simulation = NBodySimulation(lj_system, (0.0, t), pbc, _ϵ/T)

    return simulation
end
```

```
setup (generic function with 1 method)
```

In order to compare different integrating methods we will consider a fixed simulation time and change the timestep (or tolerances in the case of adaptive methods).

```julia
function benchmark(energyerr, rts, bytes, allocs, nt, nf, t, configs)
    simulation = setup(t)
    prob = SecondOrderODEProblem(simulation)
    for config in configs
        alg = config.alg
        sol, rt, b, gc, memalloc = @timed solve(prob, alg();
            save_everystep=false, progress=true, progress_name="$alg", config...)
        result = NBodySimulator.SimulationResult(sol, simulation)
        ΔE = total_energy(result, t) - total_energy(result, 0)
        energyerr[alg] = ΔE
        rts[alg] = rt
        bytes[alg] = b
        allocs[alg] = memalloc
        nt[alg] = sol.destats.naccept
        nf[alg] = sol.destats.nf + sol.destats.nf2
    end
end

function run_benchmark!(results, t, integrators, tol...; c=ones(length(integrators)))
    @progress "Benchmark at t=$t" for τ in zip(tol...)
        runtime = Dict()
        ΔE = Dict()
        nt = Dict()
        nf = Dict()
        b = Dict()
        allocs = Dict()
        cfg = config(integrators, c, τ...)

        GC.gc()
        benchmark(ΔE, runtime, b, allocs, nt, nf, t, cfg)
        get_tol(idx) = haskey(cfg[idx], :dt) ? cfg[idx].dt : (cfg[idx].abstol,
cfg[idx].rtol)

        for (idx,i) in enumerate(integrators)
            push!(results, [string(i), runtime[i], get_tol(idx)..., abs(ΔE[i]), nt[i],
nf[i], c[idx]])
        end
    end
    return results
end
```

```
run_benchmark! (generic function with 1 method)
```

We will consider symplectic integrators first

```julia
symplectic_integrators = [
    VelocityVerlet,
    #VerletLeapfrog,
    PseudoVerletLeapfrog,
    McAte2,
    #CalvoSanz4,
    #McAte5,
    Yoshida6,
    #KahanLi8,
    #SofSpa10
]
```

```
4-element Array{DataType,1}:
 VelocityVerlet
 PseudoVerletLeapfrog
 McAte2
 Yoshida6
```

Let us run a short simulation to see the cost per timestep for each method

```
config(integrators, c, τ) = [ (alg=a, dt=τ*c_a) for (a,c_a) in zip(integrators, c)]

t = 35.0
τs = 1e-3

# warmup
c_symplectic = ones(length(symplectic_integrators))
benchmark(Dict(), Dict(), Dict(), Dict(), Dict(), Dict(), 10.,
    config(symplectic_integrators, c_symplectic, τs))

results = DataFrame(:integrator=>String[], :runtime=>Float64[], :τ=>Float64[],
    :EnergyError=>Float64[], :timesteps=>Int[], :f_evals=>Int[], :cost=>Float64[]);
run_benchmark!(results, t, symplectic_integrators, τs)
```

	integrator	runtime		EnergyError	timesteps	f_evals	cost
	String	Float64	Float64	Float64	Int64	Int64	Float64
1	VelocityVerlet	58.3117	0.001	0.00366562	35000	70002	1.0
2	PseudoVerletLeapfrog	58.9367	0.001	0.0208884	35000	105002	1.0
3	McAte2	57.3776	0.001	0.00980374	35000	105002	1.0
4	Yoshida6	228.282	0.001	0.0219439	35000	525002	1.0

The cost of a timestep can be computed as follows

```
c_symplectic .= results[!, :runtime] ./ results[!, :timesteps]
c_Verlet = c_symplectic[1]
c_symplectic /= c_Verlet
```

```
4-element Array{Float64,1}:
 1.0
 1.0107192821064712
 0.9839818605163992
 3.9148652045656704
```

were we have normalized the cost to the cost of a VelocityVerlet step.

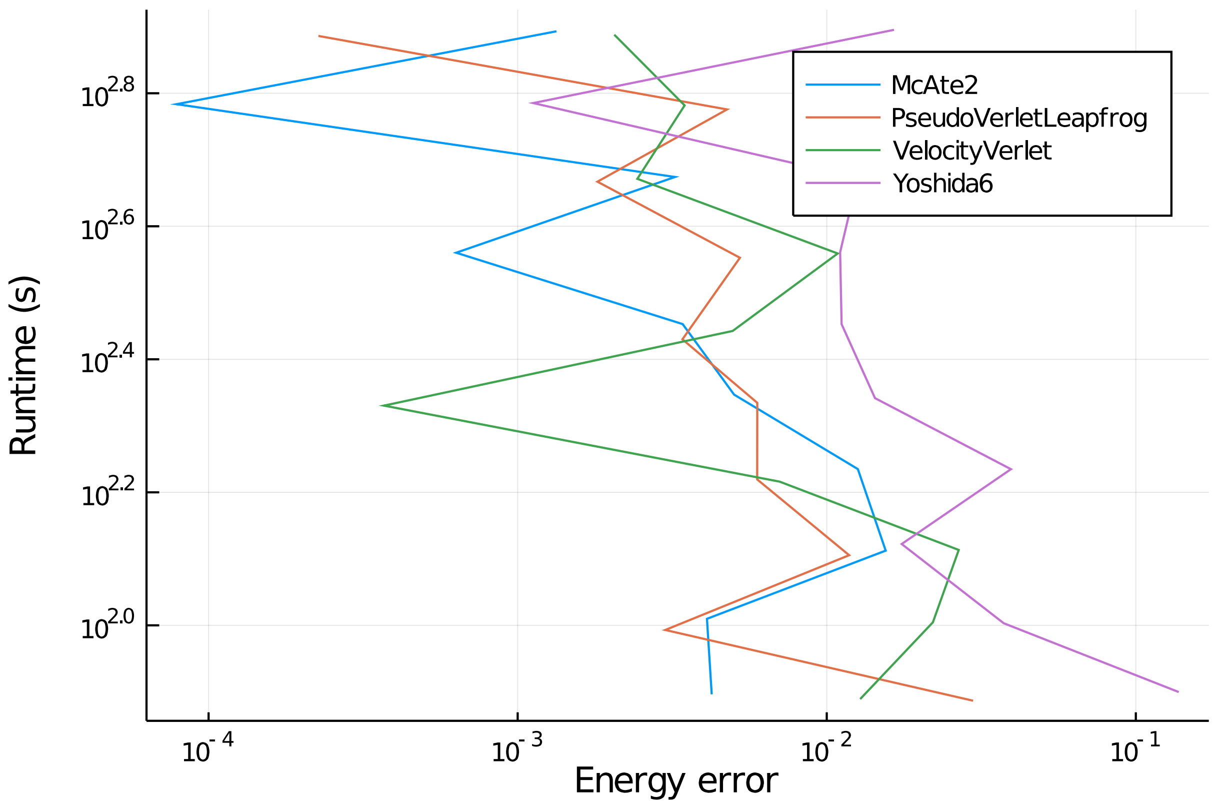
Let us now benchmark the solvers for a fixed simulation time and variable timestep

```
t = 45.0
τs = 10 .^range(-4, -3, length=10)

results = DataFrame(:integrator=>String[], :runtime=>Float64[], :τ=>Float64[],
    :EnergyError=>Float64[], :timesteps=>Int[], :f_evals=>Int[], :cost=>Float64[]);
run_benchmark!(results, t, symplectic_integrators, τs, c=c_symplectic)
```

	integrator	runtime		EnergyError	timesteps	f_evals	cost
	String	Float64	Float64	Float64	Int64	Int64	Float64
1	VelocityVerlet	772.73	0.0001	0.00205374	450000	900002	1.0
2	PseudoVerletLeapfrog	769.411	0.000101072	0.000226694	445228	1335686	1.01072
3	McAte2	781.916	9.83982e-5	0.00133502	457326	1371980	0.983982
4	Yoshida6	785.892	0.000391487	0.016506	114947	1724207	3.91487
5	VelocityVerlet	604.974	0.000129155	0.00346887	348419	696840	1.0
6	PseudoVerletLeapfrog	596.589	0.000130539	0.00475678	344724	1034174	1.01072
7	McAte2	607.519	0.000127086	7.87699e-5	354091	1062275	0.983982
8	Yoshida6	609.882	0.000505624	0.00112326	88999	1334987	3.91487
9	VelocityVerlet	469.181	0.00016681	0.00243857	269768	539538	1.0
10	PseudoVerletLeapfrog	464.585	0.000168598	0.0018116	266907	800723	1.01072
11	McAte2	472.201	0.000164138	0.00322909	274160	822482	0.983982
12	Yoshida6	470.586	0.000653039	0.0125931	68909	1033637	3.91487
13	VelocityVerlet	362.315	0.000215443	0.0108509	208872	417746	1.0
14	PseudoVerletLeapfrog	356.996	0.000217753	0.00523696	206657	619973	1.01072
15	McAte2	363.282	0.000211992	0.000632298	212272	636818	0.983982
16	Yoshida6	363.936	0.000843432	0.0110485	53354	800312	3.91487
17	VelocityVerlet	277.096	0.000278256	0.00496142	161722	323446	1.0
18	PseudoVerletLeapfrog	269.202	0.000281239	0.00341266	160007	480023	1.01072
19	McAte2	283.751	0.000273799	0.00341795	164355	493067	0.983982
20	Yoshida6	283.687	0.00108933	0.0111669	41310	619652	3.91487
21	VelocityVerlet	213.978	0.000359381	0.000369399	125216	250434	1.0
22	PseudoVerletLeapfrog	216.063	0.000363234	0.00595934	123888	371666	1.01072
23	McAte2	222.285	0.000353625	0.00501406	127254	381764	0.983982
24	Yoshida6	219.59	0.00140693	0.0143232	31985	479777	3.91487
25	VelocityVerlet	164.451	0.000464159	0.00704297	96950	193902	1.0
26	PseudoVerletLeapfrog	165.738	0.000469134	0.00595511	95922	287768	1.01072
27	McAte2	171.765	0.000456724	0.0126058	98528	295586	0.983982
28	Yoshida6	171.697	0.00181712	0.0394393	24765	371477	3.91487
29	VelocityVerlet	129.825	0.000599484	0.0267297	75065	150132	1.0
30	PseudoVerletLeapfrog	127.478	0.00060591	0.0118152	74269	222809	1.01072
31	McAte2	129.514	0.000589882	0.0155126	76287	228863	0.983982
32	Yoshida6	132.527	0.0023469	0.0174756	19175	287627	3.91487
33	VelocityVerlet	101.054	0.000774264	0.0220466	58120	116242	1.0
34	PseudoVerletLeapfrog	98.4098	0.000782563	0.00300324	57504	172514	1.01072
35	McAte2	102.285	0.000761861	0.00409616	59066	177200	0.983982
36	Yoshida6	100.733	0.00303114	0.0373672	14846	222692	3.91487
37	VelocityVerlet	77.4751	0.001	0.0128268	45001	90004	1.0
38	PseudoVerletLeapfrog	77.0364	0.00101072	0.029744	44523	133571	1.01072
39	McAte2	78.7528	0.000983982	0.00424444	45733	137201	0.983982
40	Yoshida6	79.3848	0.00391487	0.137679	11495	172427	3.91487

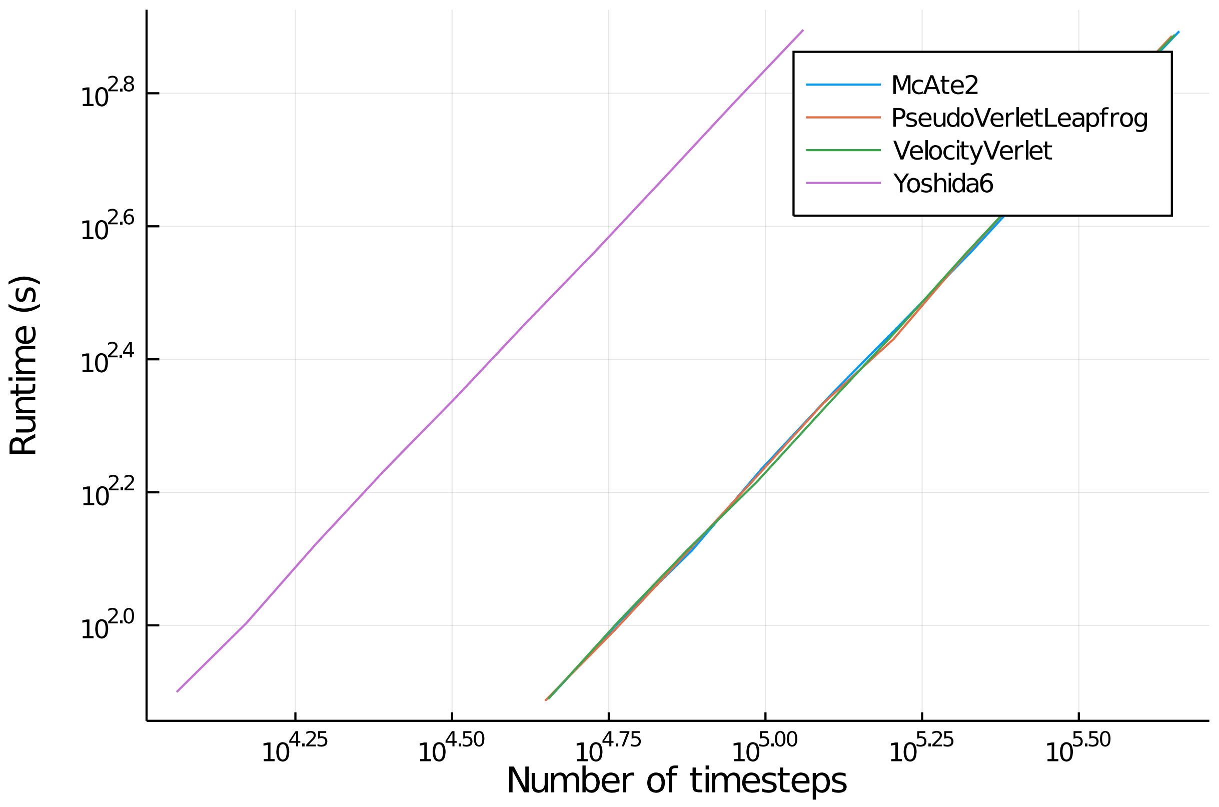
The energy error as a function of runtime is given by

```
@df results plot(:EnergyError, :runtime, group=:integrator,
    xscale=:log10, yscale=:log10, xlabel="Energy error", ylabel="Runtime (s)")
```



Looking at the runtime as a function of timesteps, we can observe that we have a linear dependency for each method, and the slope is the previously computed cost per step.

```
@df results plot(:timesteps, :runtime, group=:integrator,
    xscale=:log10, yscale=:log10, xlabel="Number of timesteps", ylabel="Runtime (s)")
```



We can also consider a longer simulation time

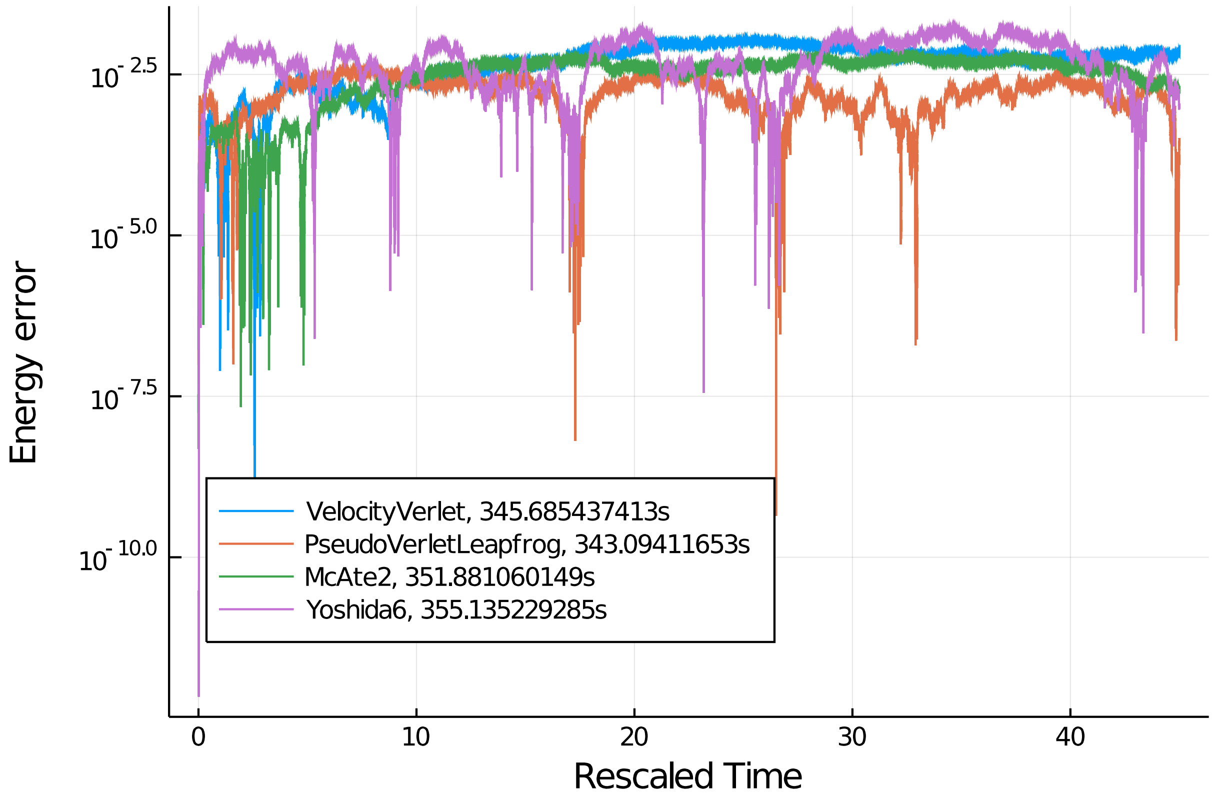
```
t = 100.0

τs = 10 .^range(-4, -3, length=5)

results = DataFrame(:integrator=>String[], :runtime=>Float64[], :τ=>Float64[],
    :EnergyError=>Float64[], :timesteps=>Int[], :f_evals=>Int[], :cost=>Float64[]);
#run_benchmark!(results, t, symplectic_integrators, τs, c=c_symplectic)
```

The energy error as a function of runtime is given by

```
#@df results plot(:EnergyError, :runtime, group=:integrator,
#    xscale=:log10, yscale=:log10, xlabel="Energy error", ylabel="Runtime (s)")
```

We can also look at the energy error history

```
function benchmark(energyerr, rts, ts, t, configs)
    simulation = setup(t)
    prob = SecondOrderODEProblem(simulation)
    for config in configs
        alg = config.alg
        sol, rt = @timed solve(prob, alg(); progress=true, progress_name="$alg",
config...)
        result = NBodySimulator.SimulationResult(sol, simulation)
        ∆E(t) = total_energy(result, t) - total_energy(result, 0)
        energyerr[alg] = [∆E(t) for t in sol.t[2:end]]
        rts[alg] = rt
        ts[alg] = sol.t[2:end]
    end
end

ΔE = Dict()
rt = Dict()
ts = Dict()
configs = config(symplectic_integrators, c_symplectic, 2.3e-4)
benchmark(ΔE, rt, ts, 45., configs)

plt = plot(xlabel="Rescaled Time", ylabel="Energy error", legend=:bottomleft);
for c in configs
    plot!(plt, ts[c.alg], abs.(ΔE[c.alg]), label="$(c.alg), $(rt[c.alg])s",
yscale=:log10)
end
plt
```



Now, let us compare some adaptive methods

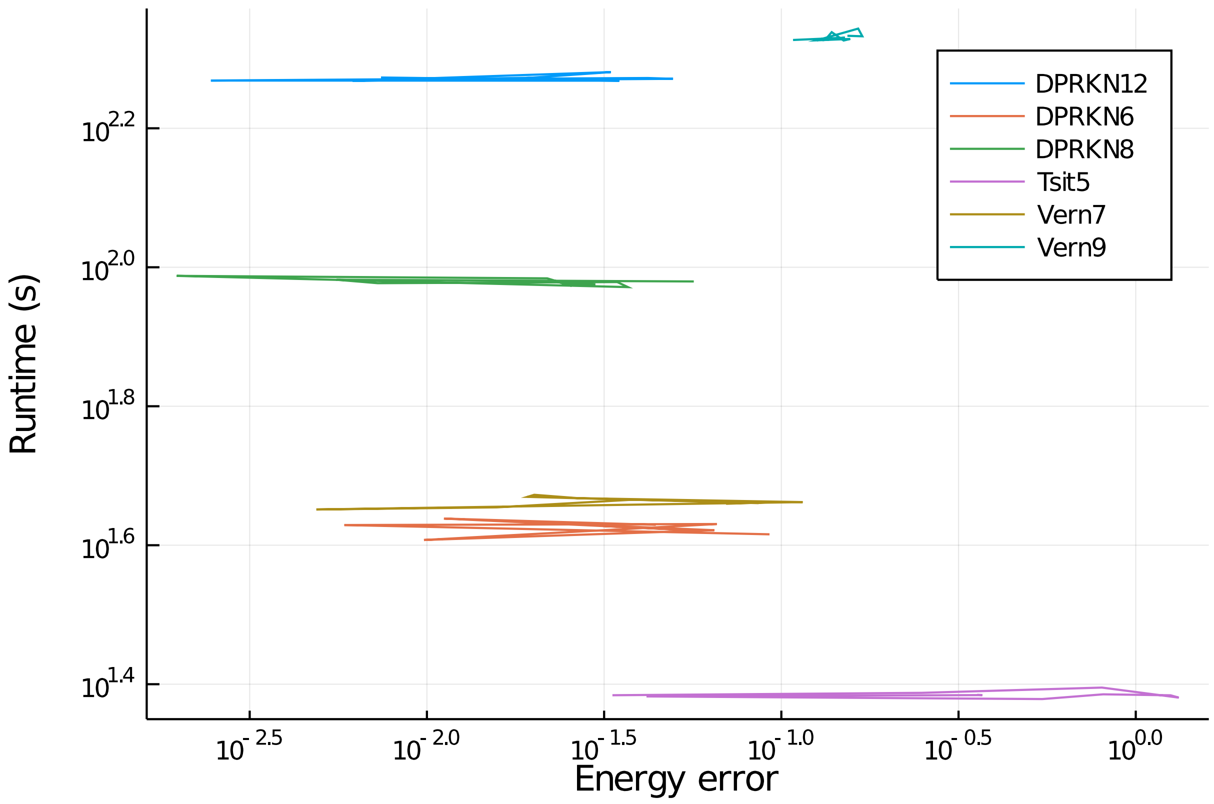
```
adaptive_integrators=[
    # DPRKN
    DPRKN6,
    DPRKN8,
    DPRKN12,
    # others
    Tsit5,
    Vern7,
    Vern9
]

config(integrators, c, at, rt) = [ (alg=a, abstol=at, rtol=rt) for a in integrators]
```

```
t = 45.0
ats = 10 .^range(-20, -14, length=10)
rts = 10 .^range(-20, -14, length=10)

# warmup
benchmark(Dict(), Dict(), Dict(), Dict(), Dict(), Dict(), 10.,
    config(adaptive_integrators, 1, ats[1], rts[1]))

results = DataFrame(:integrator=>String[], :runtime=>Float64[], :abstol=>Float64[],
    :reltol=>Float64[], :EnergyError=>Float64[], :timesteps=>Int[], :f_evals=>Int[],
:cost=>Float64[]);
run_benchmark!(results, t, adaptive_integrators, ats, rts)
```

	integrator	runtime	abstol	reltol	EnergyError	timesteps	f_evals	cost
	String	Float64	Float64	Float64	Float64	Int64	Int64	Float64
1	DPRKN6	42.5967	1.0e-20	1.0e-20	0.0442165	7661	57187	1.0
2	DPRKN8	95.4095	1.0e-20	1.0e-20	0.0566201	11837	124894	1.0
3	DPRKN12	187.534	1.0e-20	1.0e-20	0.00742181	12562	229576	1.0
4	Tsit5	24.2178	1.0e-20	1.0e-20	0.0333757	4234	28113	1.0
5	Vern7	45.7723	1.0e-20	1.0e-20	0.0862839	5123	53262	1.0
6	Vern9	212.353	1.0e-20	1.0e-20	0.107903	14377	246082	1.0
7	DPRKN6	43.4869	4.64159e-20	4.64159e-20	0.0111699	7778	58566	1.0
8	DPRKN8	95.952	4.64159e-20	4.64159e-20	0.00555696	11738	124034	1.0
9	DPRKN12	185.576	4.64159e-20	4.64159e-20	0.0348998	12581	229900	1.0
10	Tsit5	24.4096	4.64159e-20	4.64159e-20	0.249101	4206	28011	1.0
11	Vern7	46.7294	4.64159e-20	4.64159e-20	0.0192228	5147	53642	1.0
12	Vern9	213.98	4.64159e-20	4.64159e-20	0.151188	14261	244290	1.0
13	DPRKN6	41.8388	2.15443e-19	2.15443e-19	0.0561154	7604	56753	1.0
14	DPRKN8	94.8655	2.15443e-19	2.15443e-19	0.00726108	11718	123224	1.0
15	DPRKN12	185.652	2.15443e-19	2.15443e-19	0.00245804	12565	229468	1.0
16	Tsit5	24.8396	2.15443e-19	2.15443e-19	0.802825	4256	28641	1.0
17	Vern7	47.0502	2.15443e-19	2.15443e-19	0.0200828	5177	54032	1.0
18	Vern9	212.435	2.15443e-19	2.15443e-19	0.132925	14250	244322	1.0
19	DPRKN6	42.406	1.0e-18	1.0e-18	0.0319439	7676	57614	1.0
20	DPRKN8	95.2195	1.0e-18	1.0e-18	0.0344401	11755	123944	1.0
21	DPRKN12	186.379	1.0e-18	1.0e-18	0.00742968	12612	230206	1.0
22	Tsit5	24.0272	1.0e-18	1.0e-18	1.32227	4190	27909	1.0
23	Vern7	46.5304	1.0e-18	1.0e-18	0.0266389	5121	53422	1.0
24	Vern9	217.946	1.0e-18	1.0e-18	0.138736	14583	251522	1.0
25	DPRKN6	42.4787	4.64159e-18	4.64159e-18	0.0320698	7711	57824	1.0
26	DPRKN8	93.6348	4.64159e-18	4.64159e-18	0.0369092	11630	121664	1.0
27	DPRKN12	187.215	4.64159e-18	4.64159e-18	0.0420185	12620	230710	1.0
28	Tsit5	24.2123	4.64159e-18	4.64159e-18	1.2536	4176	27885	1.0
29	Vern7	45.9246	4.64159e-18	4.64159e-18	0.114867	5083	52732	1.0
30	Vern9	211.981	4.64159e-18	4.64159e-18	0.149868	14279	245106	1.0
31	DPRKN6	41.8393	2.15443e-17	2.15443e-17	0.0646966	7600	56774	1.0
32	DPRKN8	97.1912	2.15443e-17	2.15443e-17	0.00196891	11989	127554	1.0
33	DPRKN12	186.721	2.15443e-17	2.15443e-17	0.049486	12607	230332	1.0
34	Tsit5	24.2911	2.15443e-17	2.15443e-17	0.809222	4205	28077	1.0
35	Vern7	44.8161	2.15443e-17	2.15443e-17	0.00487758	4990	51762	1.0
36	Vern9	213.087	2.15443e-17	2.15443e-17	0.156428	14255	244818	1.0
37	DPRKN6	40.5086	1.0e-16	1.0e-16	0.00981089	7485	55479	1.0
38	DPRKN8	96.3938	1.0e-16	1.0e-16	0.0218561	11836	125054	1.0
39	DPRKN12	185.993	1.0e-16	1.0e-16	0.0159265	12530	228622	1.0
40	Tsit5	23.9124	1.0e-16	1.0e-16	0.54487	4222	28083	1.0
41	Vern7	45.134	1.0e-16	1.0e-16	0.0157425	5075	52762	1.0
42	Vern9	212.148	1.0e-16	1.0e-16	0.126	14267	244546	1.0
43	DPRKN6	42.7097	4.64159e-16	4.64159e-16	0.0657288	7604	56963	1.0
44	DPRKN8	94.0587	4.64159e-16	4.64159e-16	0.0256069	11605	121754	1.0
45	DPRKN12	190.929	4.64159e-16	4.64159e-16	0.0330368	12805	234778	1.0
46	Tsit5	24.1221	4.64159e-16	4.64159e-16	0.041691	4192	27795	1.0
47	Vern7	46.2898	4.64159e-16	4.64159e-16	0.0379866	5071	52972	1.0
48	Vern9	220.532	4.64159e-16	4.64159e-16	0.164754	14400	248226	1.0
49	DPRKN6	42.5632	2.15443e-15	2.15443e-15	0.00584886	7646	57033	1.0
50	DPRKN8	94.7994	2.15443e-15	2.15443e-15	0.0241828	11761	123794	1.0
51	DPRKN12	185.446	2.15443e-15	2.15443e-15	0.00616829	12585	229702	1.0
52	Tsit5	24.2282	2.15443e-15	2.15443e-15	0.368856	4179	27717	1.0

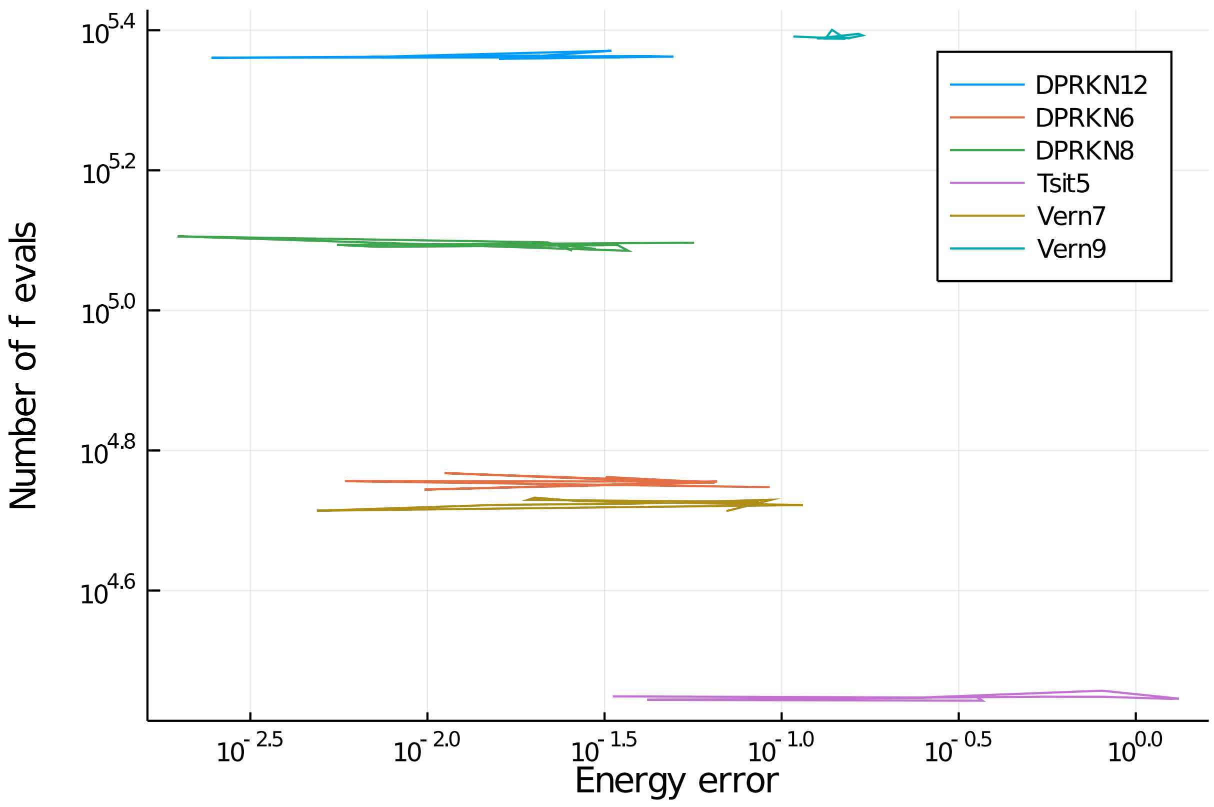
The energy error as a function of runtime is given by

```
@df results plot(:EnergyError, :runtime, group=:integrator,
    xscale=:log10, yscale=:log10, xlabel="Energy error", ylabel="Runtime (s)")
```



If we consider the number of function evaluations instead, we obtain

```
@df results plot(:EnergyError, :f_evals, group=:integrator,
    xscale=:log10, yscale=:log10, xlabel="Energy error", ylabel="Number of f evals")
```



We can also consider a longer simulation time

```
t = 100.0

ats = 10 .^range(-20, -14, length=10)
rts = 10 .^range(-20, -14, length=10)

results = DataFrame(:integrator=>String[], :runtime=>Float64[], :abstol=>Float64[],
    :reltol=>Float64[], :EnergyError=>Float64[], :timesteps=>Int[], :f_evals=>Int[],
:cost=>Float64[]);
#run_benchmark!(results, t, integrators, ats, rts)
```

The energy error as a function of runtime is given by

```
#@df results plot(:EnergyError, :runtime, group=:integrator,
#    xscale=:log10, yscale=:log10, xlabel="Energy error", ylabel="Runtime (s)")
```