Asynchronous Time Integrator With a Randomness Method

Henrique Matulis, Qiqi Xu
University of Toronto
Computer Science Department

Abstract

We reviewed past work on time integration and stability analysis. We looked into multi-time integration and probabilistic numerical integration in particular. We proposed one way of dynamically assigning time steps during integration, which can be used in an asynchronous or synchronous setting. A few examples using past assignment code are shown.

Keywords: time integrator, asynchronous, randomness

1 Introduction

People have been looking for ways to improve integration for a long time. Integration types includes implicit and explicit integration. In class assignments we integrated entire objects with a single integration type and time step for the whole object. We also saw that this was not very stable[1]. Sometimes using different types or different time steps can achieve faster and more stable integration. The former is referred to as mixed-time integration, and the latter is referred to as multi-time integration or subcycling.

In the field of mixed-time integration, [Belytschko and Mullen 1976] first introduced a way to partition the mesh into two parts and implicitly integrate one part, but explicitly integrate the other. An integrator that partitions the mesh in this way is called an explicit-implicit time integrator. An alternative way is a mixed-time integration method called explicit^m-explicit method, which used explicit method for both parts but different time steps as introduced in [T.Belytschko et al. 1979]. One drawback was that the time step ratio of adjacent parts were forced to be integers, i.e. Δt and $m\Delta t$. This constraint was relaxed by [Neal and Belytschko 1989], which is an explicit-explicit subcycling method. Later, an implicit subcycling time integration method was developed by [Smolinski and Wu 1998]. These all involve some partition mechanism, [A. Lew and West 2003] developed a general framework for asynchronous time integrator algorithms, allowing **each** elements to have a different time step, with no integer ratio requirement neither. We actually found the idea of "asynchronicity" can be applied to many other fields, for example, collision detection [Samantha et al. 2012], cloth simulation [Thomaszewski and Blochinger 2006] and real time deformations [Debunne et al. 2001].

There is also much work on the stability of these time integration methods. Perhaps the first and most important one is [Belytschko and Mullen 1977], in which they examined the stability of explicit-implicit work they proposed in 1976. They proved the the Courant condition is necessary for stability in energy (bounded energy) for linear systems in explicit integration.

For all work we found on asynchronous methods, a fixed time step is used throughout the simulation. In our work, we explore the possibility of assigning time steps dynamically depending on some heuristic measure of instability of the simulation. We use randomness to have this heuristic measure.

[1] A2 Bunny Link

2 Related Work

The first component of our work is randomness. One of the first probabilistic ODE methods is developed by [Conrad1 et al. 2016]. Their update equation is:

$$Y_{k+1} = \Phi_h(Y_k) + \psi(h)$$

where Φ is a deterministic integrator solution and ψ is a scaled i.i.d. Gaussian random variable. Later [Abdulle and Garegnani 2018] proposed a method to quantify the uncertainty induced by the time integration of ordinary differential equations (ODEs). They achieve this by introducing suitable random time-steps to integrator. i.e.

$$Y_{k+1} = \Phi_{H_k}(Y_k)$$

where H_k is a random variable for time step. Our work bare the similarity between both methods, in that we have an error term while having random time steps.

Another component of our paper is asynchronicity. The idea is closely related to mixed-time integration, and was formalized by [A. Lew and West 2003]. In their selected examples, the elemental time steps are determined from the Courant condition. They then use a priority queue to solve the problem that the elements have no guarantee to resynchronize at any time. Our work will build from their algorithm.

3 Problem Statement

3.1 Preliminaries

3.1.1 Notation

- q_p: generalized coordinate at p
- $\dot{q_p}$: generalized velocity at p
- $\ddot{q_p}$: generalized accelerations at p
- $q^{(n)}$: n-th derivative of q with respect to time
- t: present time (global time clock)
- Δt_p: time step to update element e.

3.1.2 Stability Condition

Courant–Friedrichs–Lewy condition is what we need in an explicit integration method.

3.1.3 Asynchronous update

Due to asynchronicity, the particles almost never resynchronize. To solve this, we adopt the idea from AVI: maintain a priority queue,

such that the top particle in the queue (next to be processed) is the particle whose next activation time is closest to the present time. Say we pop particle p, we compute $q_p(t + \Delta t_p), q_p(t + \Delta t_p)$ according to $q_p(t), q_p(t)$ in an explicit integration way as long as the simulation is not over. Add it back to the queue if the next time step comes before the target integration time. Note we compute Δt_p every iteration. Repeat this process until the queue is empty.

3.2 Objective

Suppose we have $\Delta t, q(t), q(t)', ...$ compute $q(t + \Delta t)$ and $q'(t + \Delta t)$

4 Methodology

4.1 Randomness

If q is nice enough, we can Taylor expand it:

$$q(t + \Delta t) = \sum_{n=0}^{\infty} \frac{f^{(n)}(t)}{n!} \Delta t^n$$

We construct $g(t)$ which is an estimator of $f(t)$. In other words:

$$\mathbb{E}[g(\Delta t)] = q(t + \Delta t)$$

In our implementation, we chose $g(t)$ by truncating $f(t)$ using a geometric random variable k, and adjusting each term. Specifically:

$$g(t) = \sum_{n=0}^{k} \frac{q^{(n)}(t)}{n!} 2^{n-2} \Delta t^n$$

with probability

$$\sum_{n=0}^{k} \frac{1}{2^{n-2}}$$

We can verify that $g(t)$ is actually an unbiased estimation of $f(t)$.

4.2 Asynchronicity

We used the priority queue idea described in section 3.1.3.

We use the variance of $g(t)$ to dynamically alter the time step of an element through the simulation.

$$\begin{aligned} Var[g(t)] &= \mathbb{E}[(g(t) - \mathbb{E}[g(t)])^2] \\ &= \mathbb{E}[(g(t) - q(t))^2] \end{aligned}$$

At a high level if the variance is higher than some thresh-hold, we reduce the time step size. And if it is lower than some thresh-hold, we increase the time step size up to a limit. These thresh-holds are predefined, and we got them by hand tuning. The resulting time step is denoted as $\Delta t'_p$.

Since our implementation uses an explicit method, the time steps are also limited by the Courant condition. Overall, a robust solution would have Δt_p be the min of Courant condition and $\Delta t'_p$. Here we outline an algorithm for our method.

5 Experiments

The first question we need to address is how to compute derivative efficiently. Our first attempt is to use automatic differentiation tools, but they were too slow, so we ended up deriving it using MATLAB, but since the complexity grows exponentially, MATLAB didn't give an answer in a reasonable time, so our maximum degree is 16.

Algorithm 1 Asynchronous Randomization Time Integration

Require: Δt
Initialize priority queue $queue$
$\Delta t_i = \Delta t$, Push $(\Delta t_i, i)$ to $queue$ for all $i \in V$
while $queue$ is not empty **do**
  y = $queue.top()$ i = y.second
  calculate var_i at vertex i
  IF $var_i > max_var$
  THEN $\Delta t_i /= 2$
  ElSE IF $var_i < min_var$
  THEN $\Delta t_i = max(\Delta t_i * 2, \Delta t)$
  calculate Courant condition
  crop Δt_i accordingly
  Get k from geometric distribution
  Calculate up to k-th derivatives
  Update q and $\dot{q}$ with $g(t)$ using Δt_i
  POP i from $queue$
  Push $(t + \Delta t_i$, i) to $queue$
end while

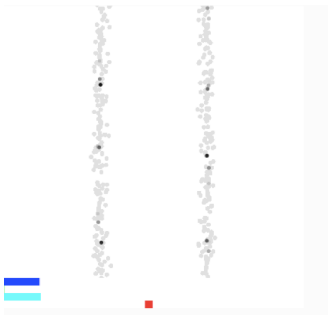

Figure 1: *A1 spring mass system*

We also use geometric distribution to get the k defined in section 4.1.

$$k = std :: min(max, min + GEO_DIST(p))$$

where max and min are maximum and minimum degrees, and bigger p means less likely to sample higher order derivatives.

We compute variance by using these first 16 derivatives. However, computing all derivatives of $q(t)$ at run-time is too expensive and does not allow us to use symbolic integration.

So we 'truncate' f's taylor series. We define a function $h(t)$:

$$h(t) = \sum_{n=0}^{16} \frac{q^{(n)}(t)}{n!} \Delta t^n$$

And then we compute the variance and update equations based on h instead of f. Note that this introduces bias.

5.1 2D Spring Mass

In figure 1, we show an example of assignment 2 using randomness only (fixed and same pre-defined time step for every particle). It's a truncated ellipse. As it shown in the picture, the points spread due to randomness, but they are close to the expectation.

5.2 3D Spring Mass

In figure 2, we show an example of assignment 2 using randomness only (fixed and same pre-defined time step for every particle). As we can see from the energy plot, the energy is actually preserved

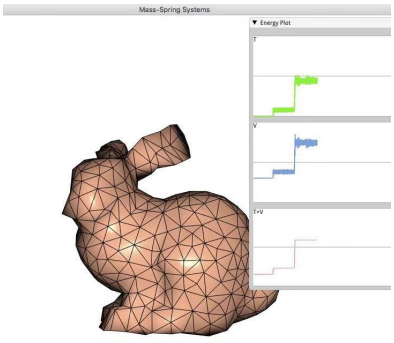

Figure 2: *Geometric distribution with probability 0.5 max = 16, min = 4*

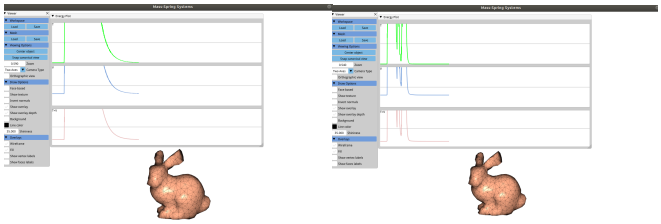

Figure 3: *air friction = 3 vs air friction = 30*

very well, since it is near constant when we don't touch the bunny. A video can be found here: A video can be found here. In this video, we can see that even if some part is unstable and collapses, other parts remain stable.

We further test this assignment on adding adaptive time steps that reflect the amount of variance induced by the randomness. In order to make the simulation more realistic, we added air friction, so energy loses as time passes. We made a few comparisons in figure 3 and the table below.

	k	min	max	air friction	adaptive
link 1	5e3	6	6	1	yes
link 2	5e3	3	6	1	yes
link 3	5e3	6	6	1	no
link 4	5e3	3	6	1	no
link 5	5e3	6	6	0.2	yes
link 6	5e3	3	6	0.2	yes
link 7	5e3	6	6	0.2	no
link 8	5e3	3	6	0.2	no

We plan to implement asynchronicity behaviour by using a priority queue.

6 Conclusion

We conclude that adding randomness to time integrator doesn't make it too wrong, and with variance as a regularization of instability, it prevents the simulation from exploding.

7 Future Work

- We didn't finish up implementing asynchronicity part, so our method uses a synchronous time step
- We didn't include stability condition in our implementation, which may be a good try in the future.
- There might be better ways to quantify variance
- There may be better way to compute derivatives
- We need to run more experiments on more complicated systems, for example, 3D finite element system, fluid system, and those combing fluid and stiff meshes systems

8 Contact Information

If you have questions or suggestions regarding this document, please contact us at "henrique.matulis@mail.utoronto.ca" or "frances.xu@mail.utoronto.ca" .

Acknowledgements

Thanks to the course CSC2549 and its assignments:)

References

A. LEW, J. E. MARSDEN, M. O., AND WEST, M., 2003. Asynchronous variational integrators. https://doi.org/10.1007/s00205-002-0212-y.

ABDULLE, A., AND GAREGNANI, G., 2018. Random time step probabilistic methods for uncertainty quantification in chaotic and geometric numerical integration. https://arxiv.org/pdf/1801.01340.pdf.

BELYTSCHKO, T., AND MULLEN, R., 1976. Mesh partitions of explicit-implicit time integrators.

BELYTSCHKO, T., AND MULLEN, R., 1977. Stability of explicit-implicit mesh partitions in time integration.

CONRAD1, GIROLAMI, SÄRKKÄ, STUART, AND ZYGALAKIS, 2016. Statistical analysis of differential equations: introducing probability measures on numerical solutions.

DEBUNNE, G., DESBRUN, M., CANI, M.-P., AND BARR, A. H., 2001. Dynamic real-time deformations using space time adaptive sampling.

NEAL, M. O., AND BELYTSCHKO, T., 1989. Explicit-explicit subcycling with non-integer time step ratios for structural dynamic systems.

SAMANTHA, A., ETIENNE, V., EITAN, G., AND RASMUS, T., 2012. Speculative parallel asynchronous contact mechanics. http://www.cs.columbia.edu/cg/spacm/spacm.html.

SMOLINSKI, P., AND WU, Y.-S., 1998. An implicit multi-time step integration method for structural dynamics problems.

T.BELYTSCHKO, H-J.YEN, AND R.MULLEN, 1979. Mixed methods for time integration. https://www-sciencedirect-com.myaccess.library.utoronto.ca/science/article/pii/0045782579900227.

THOMASZEWSKI, B., AND BLOCHINGER, W., 2006. Physically based simulation of cloth on distributed memory architectures.