Precision-dependent symmetry breaking in simulated motion of polyspherical grains

Ulrich Mutze *

A system of 10 elastic polyspherical particles, initially falling freely in the interior of a spherical cavity is simulated employing a time-stepping method and a time-stepping rate which exhibit excellent conservation of total energy. The particles are initially at rest and placed mirror-symmetrically with respect to a vertical plane that divides the spherical cavity into two halves. From the equations of mechanics it follows that this initial mirror symmetry also holds for all future states, despite the many collisions that may have happened. Accumulation of rounding errors actually destroys this behavior in a numerical simulation. This is shown by doing the computation with numerical precision 16, 20, 30, and 40.

1 Introduction

In the predominant part of computational studies in physics the representation of real numbers according to the 64-bit IEEE 754 standard is used. This standard reserves 52 binary digits (corresponding to ≈ 16 decimal digits) for representing the *significand* of a number (the remaining 12 bits represent the sign and the exponent). In the context of computing one usually refers to this circumstance by attributing the *precision* 16 (decimal) or 52 (binary) to such numbers. The numbers of precision 16 define a sufficiently fine lattice [1] within the mathematical continuum $\mathbb{R}$ that artifacts resulting from this discretization (numerical noise) become important only in very special situations [2] . This rather satisfactory status may be the reason why development of *multiple precision arithmetic* (also known as *arbitrary-precision arithmetic* or even *infinite-precision arithmetic*)

*www.ulrichmutze.de

[1] the lattice spacing near a point r is $\approx |r| \cdot 10^{-16}$

[2] This is in remarkable contrast to the situation that one had when a 32-bit representation was prevailing in scientific computing.

[12], [11], was for a long time not carried to a point where the existing tools invited usage by non-specialists. In these days, however, convenient tools are available and the present work was made possible by one such tool, named MPFRC++[13] [3] .

This tool defines within the programing language C++ the data type `mpreal` (*m*ultiple *p*recision *real*) which may replace the standard floating point type `double` in any program. A statement, say,

```
mpreal:set_default_prec(200);
```

will then provide any `mpreal`- typed number to be defined in the sequel with a storage area appropriate for a number of (binary, in this case) precision 200, or decimal [4] precision 60.

Arithmetic operations and transcendental functions will automatically be computed with an internal precision which is needed to give the above specified numerical precision for the result. Since such computation with arbitrary precision can't happen without significant loss in speed (see Figure 10), it is highly desirable to organize programs in a way that switching between fast native IEEE 754 arithmetic and arbitrary precision arithmetic is possible. The C++ language offers many possibilities [5] to achieve this.

Recently I introduced such a scheme in my C++ class system *Classes for Physics and Mathematics* (*CPM*, C+−) [7],[6], after having implemented a similar scheme for the less complex programing language Ruby [14]. With this augmentation, all *CPM*-programs can be made to work with arbitrary-precision arithmetic with virtually no additional effort. According to this scheme, switching between standard arithmetic and arbitrary precision arithmetic needs recompilation (or switching between two different executables), whereas switching within arbitrary precision from one value of the default precision to another can be done at run time. When in the following a precision value 16 is mentioned it is understood to refer to a computation done in native IEEE 754 mode, whereas all other values are realized by means of MPFRC++. This variability lets all *CPM*-programs in effect be functions of the precision parameter. Especially the results of program runs can then be considered as such functions. Then, the limit for precision tending to infinity is a self-suggesting idealization [6] which would introduce the mathematical idealization $\mathbb{R}$ in the end, without having to cope with infinitary objects along the way.

Having in mind how drastically the introduction of 64 bit floating point numbers once reduced the 'numerical problems' which were omnipresent with 32 bit numbers, one could expect that in all non-pathological problems the computational results would become effectively independent of precision if we consider storage sizes as high as, say, 128 bit or 256 bit per number. A moment's reflection shows, however, that computed

[3] which is a C++ -wrapper for a library named MPFR which, in turn, builds on a library named GMP

[4] In the following we always understand precision as decimal.

[5] e.g. switching names by suitable preprocessing

[6] Actual arbitrary precision systems assume that the number of digits can be represented as an integer of fixed length (say 64 bit). This is neither a limit built into the definition of the programing language (a well defined, well documented, and powerful formal language) nor in any way restricting practical applications.

trajectories of chaotic dynamical systems should always depend in the long term on the precision used in the computation: each call to a mathematical function occurring in the computation yields a result which differs from the correct value by some discretization error. Thus any arbitrarily selected point on a computed trajectory deviates from the exact one. When following the fate of the two points (computed and exact resp.) according to the exact evolution, we see the discrepancy growing exponentially. The computed trajectory is therefore expected to follow the exact trajectory rather closely for a while and then to diverge from it forever. When increasing the precision of the computation, we shift the point of divergence towards the future. Nevertheless, such a divergence will occur somewhere, however large the precision is selected. For a visualization one need not to know the exact trajectory: in a common visualization of the trajectories belonging to the same initial condition [7] but differing in computational precision one will see the bundle of trajectories fan out, the one of lowest precision will diverge from the common path first. The finally remaining path can be expected to represent the exact path well till the last point of divergence. We can observe this behavior in Figures 4. Further, we will consider a phenomenon in which the influence of limited precision can be seen and quantified in a single system trajectory (as opposed to several trajectories differing in precision). This phenomenon works as follows: A symmetry, actually a left-right mirror symmetry, which is obeyed by construction by the initial condition, and which is conserved by the exact dynamical law, gets lost over time as shown in Figure 8. A more explicit animated visualization of this phenomenon (related to a similar, but not identical system) can be found in [5].

2 The system under consideration

A definite system suitable for studying these phenomena by means of actual computations can be constructed as consisting of a few asymmetric extended rigid bodies (called *particles*) which move inside a spherical cavity. The interaction among particles and between particles and the enclosing wall is repulsive, elastic (i. e. it conserves total energy), and vanishes if the the subsets of space which are associated with the rigid bodies don't overlap. In addition to the elastic repulsive interaction we let the homogeneous gravity field act on the particles.

We assume that the particles fill a substantial part of the cavity and nevertheless start from a configuration in which there is no overlap between particles or particles and wall. The particles then all fall freely towards the bottom of the cavity and become engaged in collisions with other particles or with the wall. Since the particles are non-spherical, each collisions drives the particles to change their rotational state as well as their translational one. This lets the geometry of the next collision depend very sensitively upon the geometry of the previous collision. Thus the collisions effect a

[7] Practically, in more complex systems it is not easy to provide 'bit-identical' initial conditions for computer runs which differ in precision. This is so since between the common numerical input from a configuration file and the definition of the initial state there lies a computation ('call of a *constructor*' in C++ parlance) the result of which will normally depend on the precision.

cascaded amplification of small state variations. This mechanism is expected to bring chaotic motion about.

A computational model of such a system has been built by suitably configuring the *CPM*-program *PaLa* (for *pa*rticle *la*b) [1], [9], [10], which has a focus on numerical experiments with granular systems. It implements computational dynamics of polyspherical particles as described in [1] and [2]. As the title of [1] says, polyspherical particles are defined to be made of rigidly connected overlapping spherical particles.

All files that make up the program (apart from the support libraries opengl, glut, mpfr, gmp) can be downloaded from [9]. Moreover, a listing of all code is available as a single electronic document [8] which has an introductory chapter explaining the program in more general terms. (Also an executable running under Windows XP can be downloaded from [10].)

The model under consideration is very similar to the one which was used in [1] to study conservation of total energy. In our case each particle is made of three, only slightly overlapping, spheres. More spheres would increase the computational burden and stronger overlap would make the particles more spherical and the effect of collisions less dependent on the relative attitude of the collision partners. Mutual forces and torques between such particles and between particles and the cavity surface are defined in a natural manner from the mutual forces of the spherical components of the particles, for which an ansatz according to the Hertz formulas for elastic spheres under forced contact is made. All friction mechanism introduced in [1] are disabled.

The particles and their initial arrangement are shown in the stereo [8] image Figure 1. From the spherical container only a slice is shown in order to increase the visibility of the particles. As is obvious from these images there is a left-right mirror symmetry in the placement of the particles and also in their velocities, since the particles are assumed to be at rest (initially!). The same symmetry obviously holds for the container and also for the gravity field, for which the natural direction (from top to bottom) is understood.

The physical properties of the system are given here in SI units: Young's modulus of the particles and of the enclosing wall is $E = 1\,\mathrm{MPa}$ which corresponds to soft Silicon rubber. The density of particles is $\rho = 1250\,\mathrm{kg\,m^{-3}}$ which also corresponds to Silicon rubber. Each of the particles fills a part of space which is the set union of three overlapping spheres. In our case these three spherical constituents of a particle all have the same radius. For $n = 5$ particles $p_1, \ldots, p_n$ the spatial arrangement of the spherical constituents is determined by a stochastic procedure, and the x, y, z coordinates of their center-of-mass position is given by an input list, which is given here exactly in the format in which the program reads it from the basic configuration file `symmetryLost.ini`:

```
particle position list
//------------------------------------------------------------------------
   B addMirrorImages=true
   Z n=5
   Rs p1=-0.35 -0.25 0.2
   Rs p2=-0.55 -0.1 -0.1
   Rs p3=-0.15 0.0 0.1
```

[8] To be viewed with red/green or red/blue glasses (red for the left eye); this works also with a paper print-out of the image.

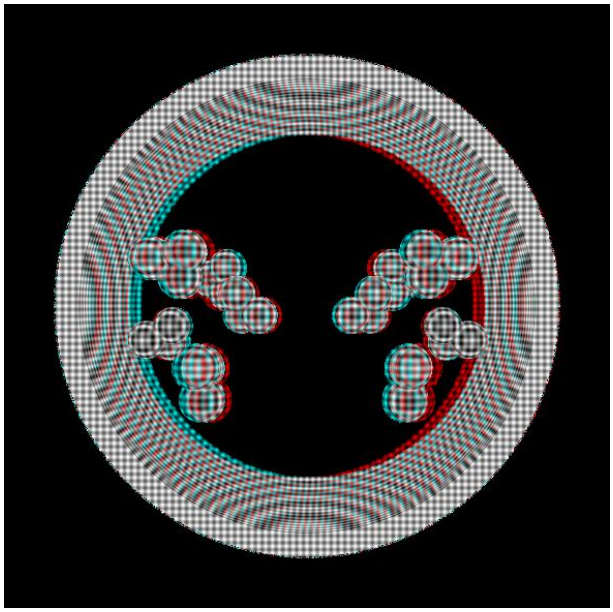


Figure 1: Initial configuration

```
Rs p4=-0.35 0.1 -0.2
Rs p5=-0.45 0.15 0.2
```

CPM configuration files organize input quantities by *sections* (here 'particle position list') and associate a *type* (here 'B,Z,Rs') and a *name* (here 'addMirrorImages, n, p1,...,p5') with each input quantity. The valid types are B (Boolean), Z (integer), R (real), W ('word', character string), and Bs, Zs, Rs, Ws for arbitrarily long lists of those. The input scheme is recursive since the full content of an arbitrarily complex configuration file may be included by a line

```
F <name of file to be included>
```

in any configuration file. Further, comment lines (starting with '/') may be placed everywhere so that a configuration file may conveniently be used as a documentation of the intents behind the program run which it defines. Since the program takes some input quantities of type W as names of files for additional input and others for output, there is much flexibility in defining a particular run of program *PaLa*.

Each particle determines a 'mean radius' as the radius of the single sphere which has the same volume as the particle (which is determined by stochastic integration). The mean radii of the particles are stochastically selected out of the interval $[0.09\,\mathrm{m}, 0.11\,\mathrm{m}]$.

The remaining five particles are made as mirror twins of the stochastically generated ones: $p_{n+i} = Rp_i$, $i = 1, \ldots, n$. Here R is the reflection with respect to the plane $x = 0$.

The radius of the cavity is $r = 0.829929\,\mathrm{m}$ and the strength of the gravity field is $a = 9.81\,\mathrm{m\,s^{-2}}$. The total span of model time of all simulation runs is $3\,\mathrm{s}$.

The computation to be reported here consists of 12 independent runs in which the precision and the total number of time-steps are varied independently over the following values:

- precisions: 16, 20, 30, 40
- time-steps: 30 000, 60 000, 120 000

These computations all have the same output: data are written to a file every 10^{-3}s of model time. Therefore we have 3000 data sets and the span between data captures is covered by 10, 20, and 40 computational steps; these are the *sub-steps* to which the legend of Figures 6, 7, and 9 refers. The data per capture are 33 numbers:

- time t
- relative energy change $(E(t)-E(0))/E(0)$
- $\log_{10}$(asymmetry measure) (see equation (1), we put $\log_{10}(0) := 0$)
- for each of the ten particles the coordinates x, y, z of the center-of-mass

Here, the asymmetry measure shown is defined as sum over the $n = 5$ mirror-pairs of particles, where each pair p_i, p_k , $k := i+n$, gives a contribution

$$d_i := \frac{\sqrt{s^2+a^2}}{r_i+r_k} \ , \quad s := |Rx_i - x_k| \ , \quad a := \varphi(r_i+r_k) \ , \tag{1}$$

where r_i is the radius of p_i, x_i its center-of-mass position, and φ is the angle of a rotation that gives p_k the same location in space (apart from a translation) as the mirrored particle Rp_i. It has obviously the value 0 whenever each particle p_i coincides with the mirror-image Rp_k of its mirror-twin p_k. It has an upper bound which is approximately n times $r_{\text{cavity}}/r_{\text{particle}}$.

3 Computational results

3.1 Trajectories of particle centers

An idea of the complex motion of the particles can be obtained from Figure 2 which shows for each of the particles the trajectory of its center-of-mass, projected onto the x-y-plane. This figure shows what was stated already: that the mirror symmetry which was built into the initial condition is conserved by dynamical evolution. To see this in spite of the detrimental influence of numerical noise one had to employ a precision of 40. (As Figure 8 shows in more detail asymmetry is also unnoticeable for precision 30, and only noticeable at the very end of the evolution for precision 20.)

What one obtains with precision 16 can be seen in Figure 3. Here we see some trajectories crossing the mirror plane $x = 0$ which violates mirror symmetry in an obvious manner. Closer inspection of this busy diagram shows that it is only at the end of the process that symmetry gets lost.

A closer look to how this asymmetry develops is hard to achieve for all particles together. So in Figure 4 we look only to particle 1 and its mirror twin particle 6:

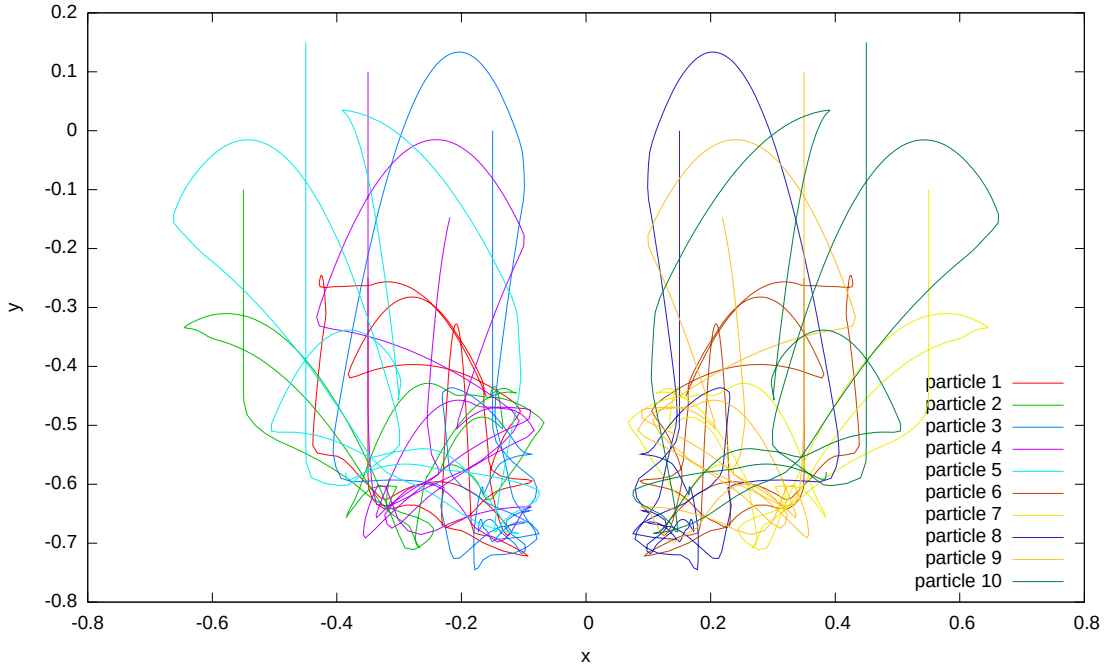


Figure 2: Trajectories of the particle centers, projected onto the x-y-plane, precision 40

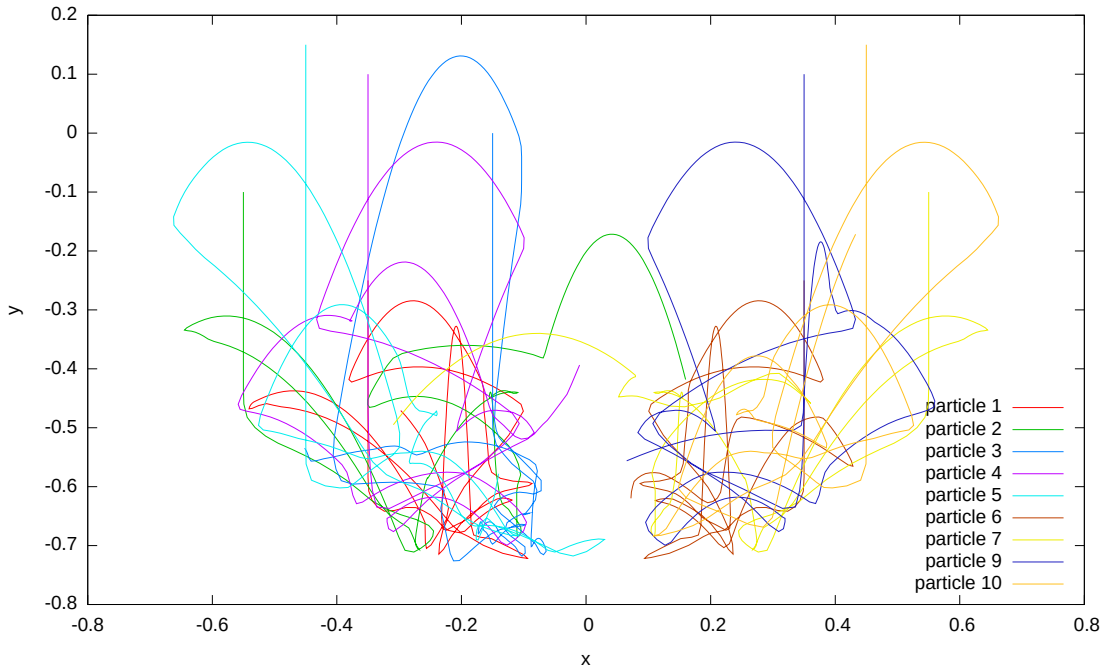


Figure 3: Trajectories of the particle centers, projected onto the x-y-plane, precision 16

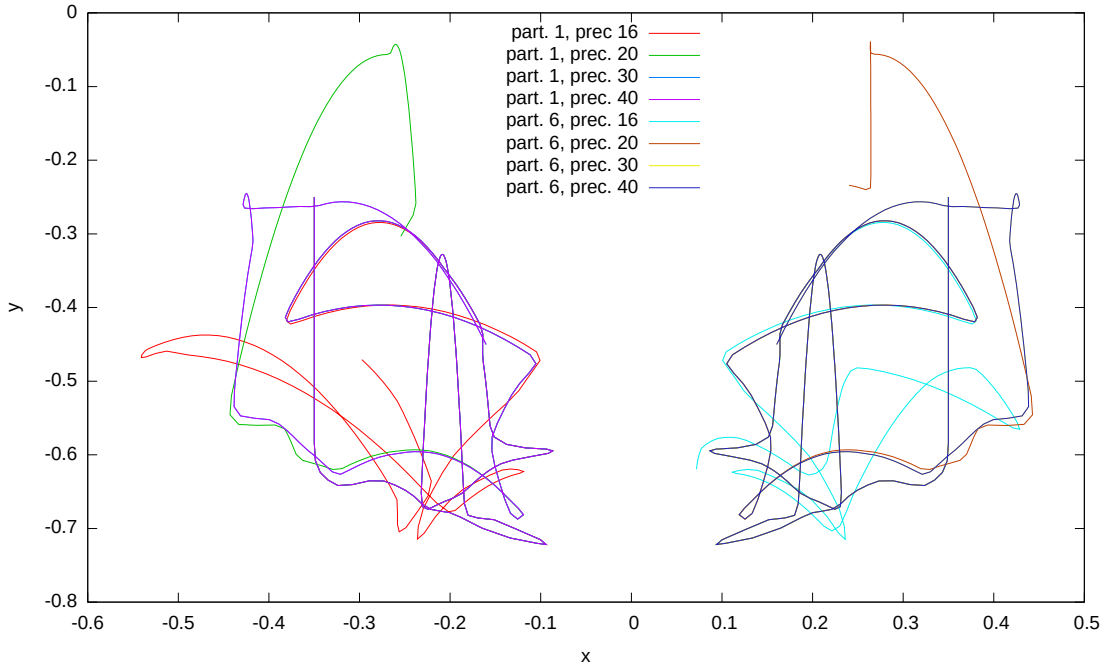


Figure 4: Trajectories of particles 1 and 6 computed with various precisions

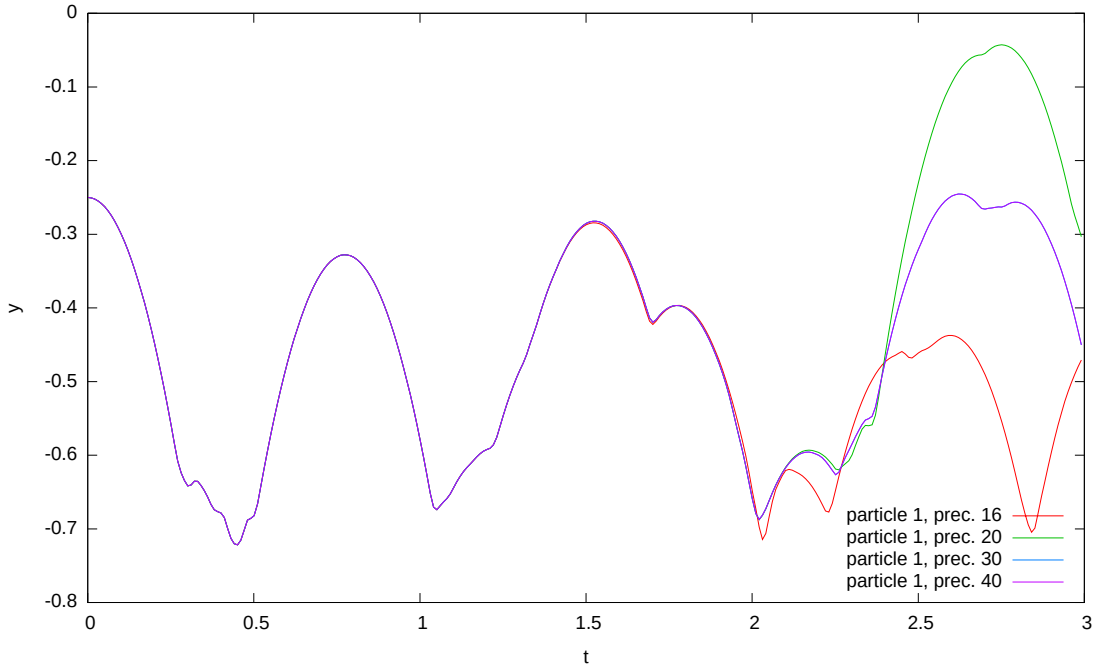


Figure 5: *y*-coordinate of particle 1 computed with various precisions

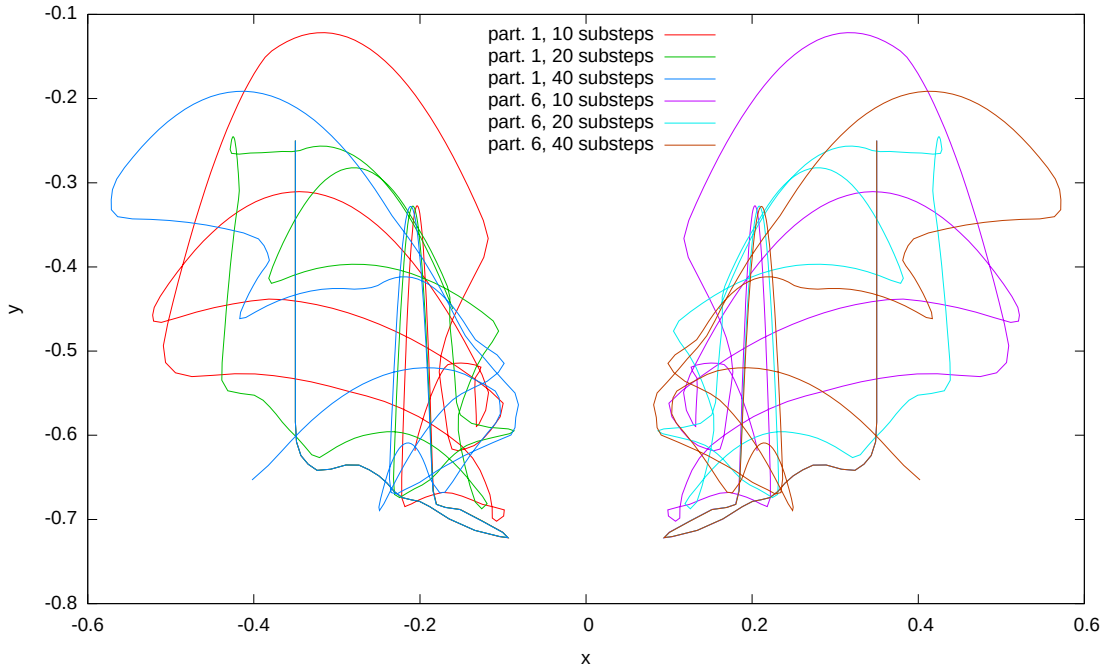


Figure 6: Trajectories of particles 1 and 6 computed with precision 40 and various time steps

The time dependence of this successive departure of less accurate trajectories is represented in Figure 5. Here the graphical resolution does not allow us to discern the curves for precision 30 and 40.

In the same format, Figure 6 shows how the variation of the time stepping rate makes the computed trajectories with lower rate deviate from those with higher rate. What we see here is that roughly the first third part of the trajectory can be expected to agree with the exact trajectory.

Notice that the obvious departure from the exact trajectory does not entail asymmetry. Actually, the time-stepping algorithm used here conserves mirror symmetry exactly when considered in $\mathbb{R}$-based arithmetic. So it is only the limited precision of the arithmetic that brings about the observed symmetry breaking.

Again, the time dependence of this successive departure of less accurate trajectories is represented in Figure 7.

3.2 Evolution of the asymmetry measure

From Figure 8 is is clear that extending the time span will finally show symmetry breaking also for precision 40. Actually, for any value of precision, there is a time span within which symmetry will be broken. Since, as stated earlier, the exact trajectory remains symmetric forever, any breakdown of symmetry shows that the computed trajectory

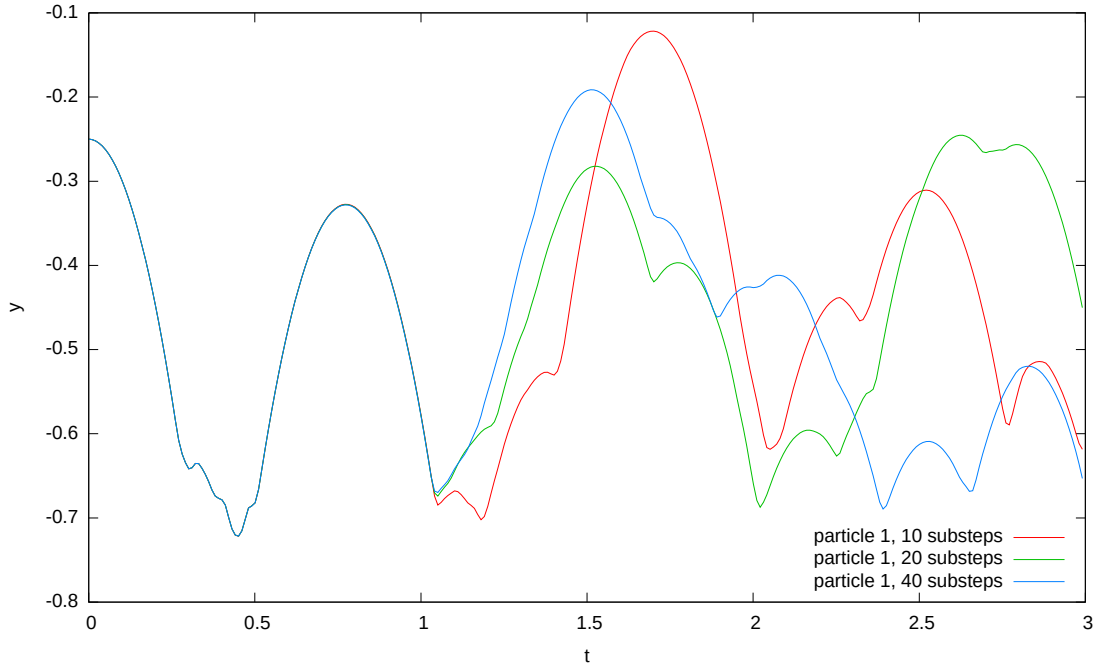


Figure 7: y-coordinate of particle 1 computed with precision 40 and various time steps

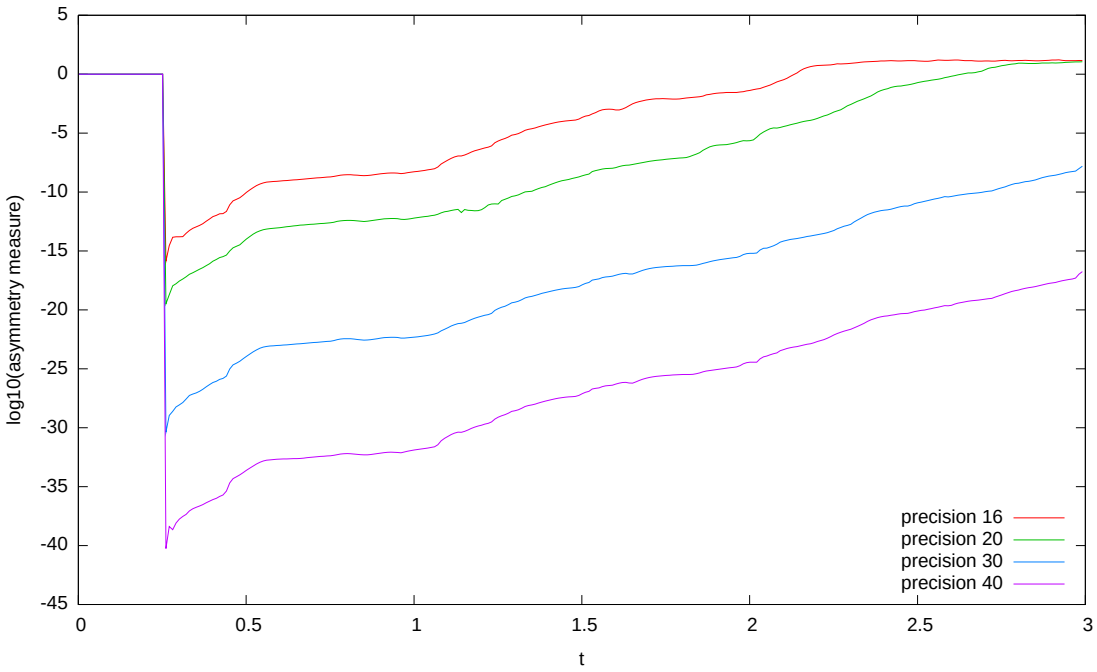


Figure 8: Evolution of the asymmetry measure computed with 40 sub-steps

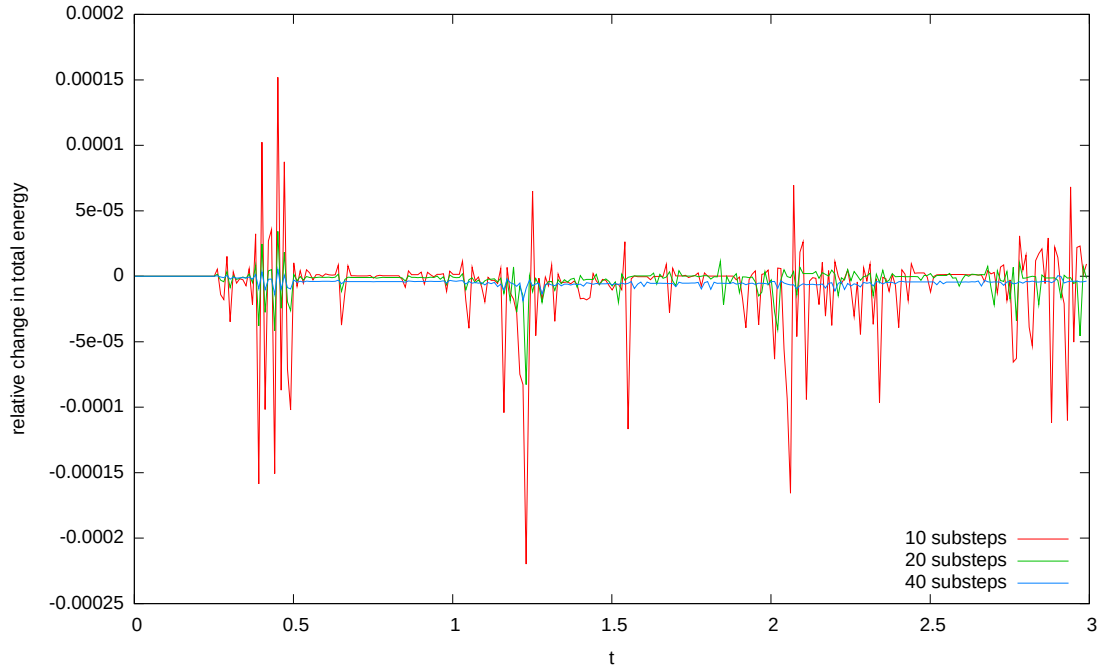


Figure 9: Change of total energy computed with precision 40 for various time steps

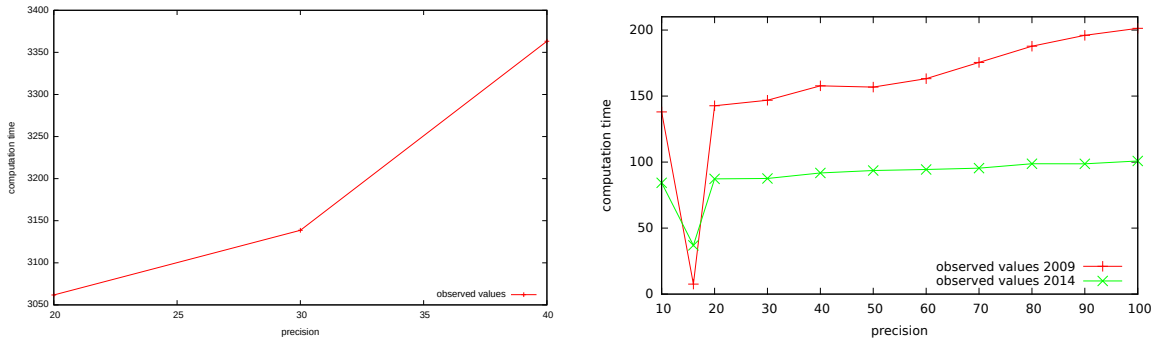
now differs from the exact trajectory. The converse, however is not true: the computed (time-discrete) trajectory may remain symmetric even if it shows no longer any similarity with the exact trajectory. An interesting feature of these curves is that the asymmetry measure, which starts with value 0 holds this value for a while till some kind of 'numerical accident' which gives a first value different from 0. It was observed in other runs that in the early phase the curve returned several times back to 0 before it turned to the normal behavior the pattern of which is obvious from the figure. When doing the same computation with simple spherical particles, I found this initial phase of vanishing asymmetry much longer, ≈ 230 s.

3.3 Evolution of total energy

The behavior of the total energy shown in Figure 9 indicates that the time-stepping rates were chosen adequately: the local variations (for which a qualitative explanation is proposed in [1]) are much larger than a hypothetical trend.

3.4 Computation time

Figure 10 shows the computation times (in seconds) needed to generate the data for varying precision and the maximum value 120 000 of time steps (which corresponds to 40 sub-steps). The speed of multiple precision computations depends surprisingly little on the precision but is much lower than the speed of precision 16 computation. The



(a) actual computation (b) test computation

Figure 10: Computation time as a function of precision

latter needed 28.875 s and thus was by a factor 106 faster than the multiple precision computation with precision 20. The test computation refers to the same system but to a shorter total time span (0.5 s instead of 3 s) so that the total computation time (for 5000 time steps) remained moderate. Also a part of the file-writing activity was disabled in order to capture mainly the speed of the genuine computation.

> Addition 2014-02-18: The original data were created under 32 bit Windows XP using a Microsoft compiler and the new data under Ubuntu-Linux using g++ compilation and the most recent ubuntu installation for MPFRC++. By reasons that I did not investigate, using type double (precision 16) is much slower now and using type mpreal became by a factor ≈ 2 faster. In the test computation the speed advantage of precision 16 over mpreal precision 20 was not found to be so extreme as given by the factor 106 reported above. It was $142.656/7.547 = 18.9$ and is now only $87.3109/36.9219 = 2.36$.

4 Outlook

Simulation of the dynamics of systems of irregularly shaped bodies, as a method of studying the behavior of granular matter, does not pretend to produce reasonable approximations to exact solutions of the underlying equations of motion. For the solution of engineering problems (e.g. [3], [4]) it is not the individual trajectory which counts but statistical properties of a large collection of trajectories. The criteria according to which simulation results are accepted as saying something useful about engineering systems can normally not be formulated without reference to properties and behavior of these systems. This implies that there is no need in such situations to compute the trajectories with multiple precision: This would only waste computation time and provide a tiny progress in an illusory direction (towards producing 'exact' trajectories).

Nevertheless, from a sportive point of view, it is tempting to try 'compute the exact

trajectory' over the full range of 3 seconds for our system. This would amount at finding a realm of time-stepping rates and numerical precisions for which the computed trajectory can be shown to be independent of these parameters when judged according to the resolution of the present graphics. Notice that for the first second, as Figure 7 suggests, we probably have a solution to this problem already with the data reported here. My experiments done so far show that a further duplication of the time-stepping rate brings little progress (no noticeable difference between precisions 40 and 50), so that one has probably to go to much smaller time steps and may soon transcend comfortable computation times.

Addition 2014-02-18: Even at precision 512 the trajectories for 160 sub-steps and for $\frac{4}{3} \cdot 160$ sub-steps diverge visibly at the end of the first second.

One doesn't go into chaos comfortably!

Acknowledgment

I thank Shigeo Kobayashi and Roger Pack for furthering comments on my multiple precision Ruby work, and Roger Pack for making me aware of the library MPFR. Further I thank Pavel Holoborodko for his great tool MPFRC++, which allowed me to realize long-standing projects.

References

[1] Ulrich Mutze: Rigidly connected overlapping spherical particles: a versatile grain model, Granular Matter Vol 8, Numbers 3-4, pp. 185-194 (2006)

[2] Ulrich Mutze: Polyspherical grains and their dynamics
Mathematical Physics Preprint 07–252, July 2007
www.ma.utexas.edu/mp_arc/c/07/07-252.pdf (2007) and
www.ulrichmutze.de/articles/07-252.pdf

[3] Ulrich Mutze: Modeling the Toning Process in Electrophotographic Copiers / Printers
www.ulrichmutze.de/talks/ica1.pdf

[4] Press release of Cornell University, Center for Advanced Computing. Engineering breakthrough: a research collaboration pays off for Kodak.
http://www.cac.cornell.edu/about/studies/Kodak.pdf
http://www.tc.cornell.edu/ctc-hpc/papers/kodak.pdf

[5] Ulrich Mutze: Progressive loss of mirror symmetry in the motion of poly-spherical particles
www.ulrichmutze.de/symmetry_lost_movie/symmetry_lost.html

[6] Ulrich Mutze: Source Listing for C+−
www.ulrichmutze.de/softwaredescriptions/cpmlisting.pdf

[7] Ulrich Mutze: On the C+− project
www.ulrichmutze.de/softwaredescriptions/cpmProject.pdf

[8] Ulrich Mutze: Source Listing for C+− Program PaLa
www.ulrichmutze.de/softwaredescriptions/pala_listing.pdf

[9] Ulrich Mutze: Program PaLa (particle lab)
www.ulrichmutze.de/sourcecode/pala.zip

[10] Ulrich Mutze: Program PaLa, Windows executable
www.ulrichmutze.de/winExe/pala_distribution.zip

[11] Hiroshi Fujiwara, Yuusuke Iso: Application of multiple-precision arithmetic to direct numerical computation of inverse acoustic scattering
http://www.iop.org/EJ/article/1742-6596/73/1/012007/jpconf7_73_012007.pdf

[12] David H. Bailey, Jonathan M. Borwein: High-Precision Computation and Mathematical Physics
http://crd.lbl.gov/~dhbailey/dhbpapers/dhb-jmb-acat08.pdf

[13] Pavel Holoborodko: MPFRC++
http://www.holoborodko.com/pavel

[14] Ulrich Mutze: Applied Mathematics
https://rubyforge.org/projects/appmath/

Last modification: 2014-02-18