Note on iteration with a time-step

In computer simulations and numerical problems a frequent issue is when to end an iteration loop. The loop should run from a time `t0` to a time `t1` with a time increment `dt` (all floating point values).

A common mistake is to code this something like:

```
for (t = t0; t <= t1; t += dt) {do_whatever()}
```

You just cannot be sure if the final value for `t` in the loop will be `t1`, or more like `(t1 - dt)`.

The only solution is to somehow adapt `dt`, and there are various ways of doing that. In the following we will assume that reaching `t1` is essential and that it is not very important if a slightly larger, or a significantly smaller `dt` is used. We'll also assume that `t1 > t0`.

Basically, there are two possible approaches. The first assumes that dt will be constant throughout the loop, the second can also be used when dt is variable.

Approach 1

Precompute the number of steps required, then adapt `dt` to fit to `(t1 - t0)`:

```
Nsteps = ceil((t1 - t0) / dt);
dt = (t1 - t0) / Nsteps;
for (j = 0; j < Nsteps; j++)
{
   …;  t = t0 + j * dt; …
}
```

Note: using `t = t0 + j * dt` is generally more accurate than using `t += dt`. Why?

Approach 2

```
stop = 0;  t = t0;   epsilon = 1.0e-4 /* for example */;
while (! stop)
{
    … ; dt_max = compute_dt_max();
    if (t + (1 + epsilon) * dt_max > t1){
        dt = t1 - t; stop = 1; } else {
        dt = dt_max; }
    …
}
```

Approach 2a

Sometimes the value `dt_max` should not be exceeded at all. In such cases, it may be useful to use two half time-steps at the end, rather than a slightly stretched time-step.