FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

CAC assignment 1

Christian Møllgaard

dpj482

21. februar 2016

1 Changing the data structur

Before the problem can be vectorised, the data structure containing the data needs to be vectorised. To make the best use of memory cahing i create my solar system and asteroids as two dimensional vectors, (7,n). They have the dimension 7 first, so the values used at the same time, gets cached together.

As an example i show the new definition of the solar system array.

```
solarsystem = numpy.empty((7, n + 1))
solarsystem[:, 0] = numpy.array([1e6 * solarmass, 0, 0, 0, 0, 0, 0])
pos = numpy.random.rand(3, n)
pos[2] *= .01
dist = 1.0 / numpy.sqrt(numpy.sum(pos ** 2, axis=0)) - (.8 - numpy.random.rand(n) *
pos = maxVals * pos * dist * numpy.sign(.5 - numpy.random.rand(3, n))
magv = circlev(pos[0],pos[1],pos[2])
absangle = numpy.arctan(numpy.abs(pos[1] / pos[0]))
thetav = pi / 2 - absangle

solarsystem[1:4, 1:] = pos
solarsystem[0, 1:] = numpy.random.random(n) * solarmass * 10 + 1e20
solarsystem[4, 1:] = -1 * numpy.sign(pos[1]) * numpy.cos(thetav) * magv
solarsystem[5, 1:] = numpy.sign(pos[0]) * numpy.sin(thetav) * magv
solarsystem[6, 1:] = 0
```

2 What to vectorize

The costly action in this script is the calc force function. This function is called on every object once for every planet there is using for loops.

The main vetorizition goal is to remove these for loops. First challenge is to calculate the distance from every object to every other objet. This is done using “newaxis”. Then i need to fill the diagonal when comparing distance of planets to all planets, to make sure, there is no zero distance from planets to it self. I fill with the value 1, which is fine bacause the equation will be zero at the second half. Then i calculate the mass difference like for every planet using newaxis. Then i update the force of each planet for each dimension once at a time. This could be done using a one liner, but i found no gain from doin so, and this is easier to understand.

My calc force and move function ended up looking like this:

```
def calc_force(a, b, dt, solar):
    """Calculate forces between bodies
    F = ((G m_a m_b)/r^2)*((x_b-x_a)/r)
    """
    diff = b[1:4, numpy.newaxis, :] - a[1:4, :, numpy.newaxis]
    r = numpy.sum(diff ** 2, axis=0) ** 0.5
    if solar:
        r[numpy.diag_indices(r.shape[0])] = 1

    mass = G * b[0, numpy.newaxis, :] * a[0, :, numpy.newaxis] / (r ** 2)

    a[4] += numpy.sum(mass * (diff[0] / r) / a[0, :, numpy.newaxis] * dt, axis=1)
    a[5] += numpy.sum(mass * (diff[1] / r) / a[0, :, numpy.newaxis] * dt, axis=1)
    a[6] += numpy.sum(mass * (diff[2] / r) / a[0, :, numpy.newaxis] * dt, axis=1)



def move(solarsystem, asteroids, dt):
    """Move the bodies
    first find forces and change velocity and then move positions
    """

    calc_force(solarsystem, solarsystem, dt, True)
    calc_force(asteroids, solarsystem, dt, False)

    solarsystem[1:4] += solarsystem[4:7] * dt
    asteroids[1:4] += asteroids[4:7] * dt
```

3 Result

Running the script on marge using the config file provided, i got these results:

Tabel 1: My caption

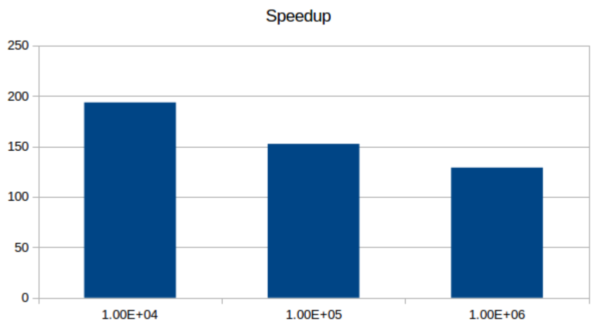
bodies	run 1	run 2	run 3	avg
1.00E+04	0.0949919224	0.0790441036	0.0791590214	0.0843983491
1.00E+05	1.1335251331	1.0382509232	1.0500969887	1.0739576817
1.00E+06	13.1345989704	12.4405839443	12.4634170532	12.6795333226

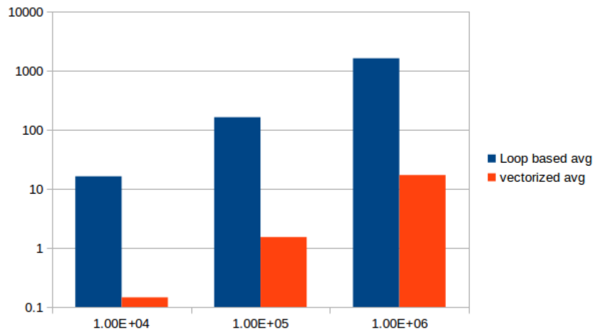
The results provided for the sequential version:

Tabel 2: My results

bodies	run 1	run 2	run 3	avg
1E4	16,21	16,49	16,25	16,32
1E5	161,89	165,88	163,16	163,65
1E6	1616,78	1658,22	1627,89	1634,30

Comparing the results of these two functions it's easy to see that there is a significant speed up





using vectors.