Generalized Symplectic Methods

The general form of a symplectic scheme to solve the conservative second-order dynamical system

$$\begin{aligned} \frac{dx}{dt} &= v \\ \frac{dv}{dt} &= a(x) \end{aligned}$$

is

$$\begin{aligned} v &= v + d_0\, a(x)\, \delta t \\ x &= x + c_0\, v\, \delta t \\ &\vdots \\ v &= v + d_{s-1}\, a(x)\, \delta t \\ x &= x + c_{s-1}\, v\, \delta t \end{aligned}$$

This generalized "kick–drift" method is defined by s, the number of stages, and the coefficients c_i and d_i. Thus, for example, the 2nd-order kick-drift-kick scheme:

```
def kdk_step(acc, t, pos, vel, dt):
    vel += 0.5*acc(pos)*dt
    pos += vel*dt
    vel += 0.5*acc(pos)*dt
    t += dt
    return t, pos, vel
```

can be rewritten as

```
c2 = np.array([1.0, 0.0])
d2 = np.array([0.5, 0.5])
def kdk_step2(acc, t, pos, vel, dt):
    for cc,dd in zip(dt*c2, dt*d2):
        vel += dd*acc(pos)
        pos += cc*vel
    t += dt
    return t, pos, vel
```

Note the use of Python's `zip` function to select pairs of variables from two `numpy` arrays.

The advantage is that, once the c and d arrays are filled in, implementing a new scheme becomes very simple. For example, here is a 3rd-order scheme (courtesy Wikipedia):

```
c3 = np.array([2./3, -2./3, 1.0])
d3 = np.array([7./24, 3./4, -1./24])
def symp_step3(acc, t, pos, vel, dt):
    for cc,dd in zip(dt*c3, dt*d3):
        vel += dd*acc(pos)
        pos += cc*vel
    t += dt
    return t, pos, vel
```

Here is an 8th-order scheme due to Yoshida (1990). The c and d arrays can become quite complicated, but the actual steps are the same in each case.

```
c8 = np.array([ 0.45742212311487,    0.5842687913979845, -0.5955794501471254,
               -0.8015464361143615,  0.8899492511272584, -0.011235547676365,
               -0.9289051917917525,  0.9056264600894919,  0.9056264600894919,
               -0.9289051917917525, -0.011235547676365,   0.8899492511272584,
               -0.8015464361143615, -0.5955794501471254,  0.5842687913979845,
                0.45742212311487 ])
d8 = np.array([ 0.,                  0.91484424622974,    0.253693336566229,
               -1.44485223686048,   -0.158240635368243,   1.93813913762276,
               -1.96061023297549,    0.102799849391985,   1.7084530707869987,
                0.102799849391985,  -1.96061023297549,    1.93813913762276,
               -0.158240635368243,  -1.44485223686048,    0.253693336566229,
                0.91484424622974 ])
def symp_step8(acc, t, pos, vel, dt):
    for cc,dd in zip(dt*c8, dt*d8):
        vel += dd*acc(pos)
        pos += cc*vel
    t += dt
    return t, pos, vel
```