

```

1  #include "ode.h"
2  void euler_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
3  {
4      uint dim = yt->dim;
5
6      uint i;
7
8      assert(dim == yt1->dim);
9
10     // evaluate function at point 't', current vector 'yt' into 'yt1'
11     // compute the right hand side function and store the value in the vector yt1
12     ---> Phi
13     f(t, yt, yt1, data);
14
15     // update new vector 'yt1'
16     for (i = 0; i < dim; i++)
17     {
18         yt->x[i] = yt->x[i] + delta * yt1->x[i];
19     }
20
21 void runge_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void *data)
22 {
23     uint dim = yt->dim;
24
25     pvector ym;
26     uint i;
27
28     assert(dim == yt1->dim);
29
30     ym = new_vector(dim);
31
32     // evaluate function at point 't', current vector 'yt' into 'yt1'
33     f(t, yt, yt1, data);
34
35     // update new vector 'ym'
36     for (i = 0; i < dim; i++)
37     {
38         ym->x[i] = yt->x[i] + 0.5 * delta * yt1->x[i];
39     }
40
41     // evaluate function at point 't+delta/2', current vector 'ym' into 'yt1'
42     f(t, ym, yt1, data);
43
44     // update new vector 'yt'
45     for (i = 0; i < dim; i++)
46     {
47         yt->x[i] = yt->x[i] + delta * yt1->x[i];
48     }
49
50     del_vector(ym);
51 }
52
53 void leapfrog_step(real t, pvector yt, ode_func f, real delta, pvector yt1, void
54 *data)
55 {
56     uint dim = yt->dim;
57     real c_mass_spring = *((real *)data);
58
59     (void)f;
60     (void)yt1;
61
62     assert(dim == yt1->dim);
63     assert(dim == 2);
64
65     yt->x[0] = yt->x[0] + 2.0 * delta * yt->x[1];
66     yt->x[1] = yt->x[1] - 2.0 * delta * c_mass_spring * yt->x[0];
67 }
68 void crank_nicolson_step(real t, pvector yt, ode_func f, real delta, pvector yt1,
69 void *data)
70 {
71     uint dim = yt->dim;

```

```
71     real c_mass_spring = *((real *)data);
72
73     uint i;
74     real onepdelta, onemdelta;
75
76     (void)f;
77
78     assert(dim == yt1->dim);
79     assert(dim == 2);
80
81     onepdelta = 1.0 + 0.25 * delta * delta * c_mass_spring;
82     onemdelta = 1.0 - 0.25 * delta * delta * c_mass_spring;
83
84     yt1->x[0] = onemdelta * yt->x[0] + delta * yt->x[1];
85     yt1->x[1] = onemdelta * yt->x[1] - delta * c_mass_spring * yt->x[0];
86
87     yt->x[0] = yt1->x[0] / onepdelta;
88     yt->x[1] = yt1->x[1] / onepdelta;
89 }
90
```