

```

1 import os, sys
2 sys.path.append(os.path.join(os.path.dirname(__file__), '../ch02'))
3 sys.path.append(os.path.join(os.path.dirname(__file__), '../ch03/program3_2'))
4
5 from program2_1 import Dvector
6 from program2_2 import Dmatrix
7 from program2_8 import vector_norm1
8 from program3_2 import gauss
9
10 EPS = 10.0 ** -8.0 # epsilon の設定
11 KMAX = 100         # 最大反復回数
12 N = 3              # N元方程式
13
14 def main():
15     print("初期値 x0, y0, z0 を入力してください---> x0 y0 z0")
16     x, y, z = map(float, input().split())
17
18     newton2( x, y, z ) # Newton法
19
20 # Newton法
21 def newton2(x: float, y: float, z: float):
22     global N
23     k = 0
24     d = Dvector(1, N)      # d[1...N]
25     xk = Dvector(1, N)     # xk[1...N]
26     J = Dmatrix(1, N, 1, N) # 行列 J[1...N][1...N]
27
28     xk[1] = x
29     xk[2] = y
30     xk[3] = z
31
32     while True:
33         # 右辺ベクトルの作成
34         d[1] = -f(xk[1], xk[2], xk[3])
35         d[2] = -g(xk[1], xk[2], xk[3])
36         d[3] = -h(xk[1], xk[2], xk[3])
37
38         # ヤコビ行列の作成
39         J[1][1] = f_x(xk[1], xk[2], xk[3])
40         J[1][2] = f_y(xk[1], xk[2], xk[3])
41         J[1][3] = f_z(xk[1], xk[2], xk[3])
42         J[2][1] = g_x(xk[1], xk[2], xk[3])
43         J[2][2] = g_y(xk[1], xk[2], xk[3])
44         J[2][3] = g_z(xk[1], xk[2], xk[3])
45         J[3][1] = h_x(xk[1], xk[2], xk[3])
46         J[3][2] = h_y(xk[1], xk[2], xk[3])
47         J[3][3] = h_z(xk[1], xk[2], xk[3])
48         d = gauss( J, d, N ) # 連立一次方程式を解く
49         for i in range(1, N+1):
50             xk[i] += d[i]
51         k += 1
52
53         if vector_norm1(d) <= EPS or k >= KMAX:
54             break
55
56     if k == KMAX:
57         print("答えが見つかりませんでした")
58     else:
59         print(f"答えは x={xk[1]}, y={xk[2]}, z={xk[3]} です")
60
61 def f(x: float, y: float, z: float) -> float:
62     return -1.0 - x + x*x - y + y*y + x*z
63
64 def g(x: float, y: float, z: float) -> float:
65     return -4.0 + 3.0*x*x + x*x*x - 2.0*z + 2.0*y*z - z*z + z*z*z
66
67 def h(x: float, y: float, z: float) -> float:
68     return -2.0*x - 3.0*y + 3.0*x*y - 4.0*z + 2.0*x*z + 4.0*y*z
69
70 def f_x(x: float, y: float, z: float) -> float:
71     return -1.0 + 2.0*x + z
72
73 def f_y(x: float, y: float, z: float) -> float:
74     return -1.0 + 2.0*y
75
76 def f_z(x: float, y: float, z: float) -> float:
77     return x
78
79 def g_x(x: float, y: float, z: float) -> float:
80     return 6.0*x + 3.0*x*x
81
82 def g_y(x: float, y: float, z: float) -> float:
83     return 2.0*z
84
85 def g_z(x: float, y: float, z: float) -> float:
86     return -2.0 + 2.0*y - 2.0*z + 3.0*z*z
87
88 def h_x(x: float, y: float, z: float) -> float:
89     return -2.0 + 3.0*y + 2.0*z
90
91 def h_y(x: float, y: float, z: float) -> float:
92     return -3.0 + 3.0*x + 4.0*z
93
94 def h_z(x: float, y: float, z: float) -> float:
95     return -4.0 + 2.0*x + 4.0*y
96
97 if __name__ == "__main__":
98     main()

```