```python
import os, sys
sys.path.append(os.path.join(os.path.dirname(__file__), '../ch02'))
sys.path.append(os.path.join(os.path.dirname(__file__), '../ch03/program3_2'))

from program2_1 import Dvector
from program2_2 import Dmatrix
from program2_8 import vector_norm1
from program3_2 import gauss

EPS  = 10.0 ** -8.0 # epsilon の設定
KMAX = 100          # 最大反復回数
N    = 3            # N元方程式

def main():
    print("初期値 x0, y0, z0 を入力してください\---> x0 y0 z0")
    x, y, z = map(float, input().split())

    newton2( x, y, z ) # Newton法


# Newton法
def newton2(x: float, y: float, z: float):
    global N
    k = 0
    d = Dvector(1, N)       # d[1...N]
    xk = Dvector(1, N)      # xk[1...N]
    J = Dmatrix(1, N, 1, N) # 行列 J[1...N][1...N]

    xk[1] = x
    xk[2] = y
    xk[3] = z

    while True:
        # 右辺ベクトルの作成
        d[1] = -f(xk[1], xk[2], xk[3])
        d[2] = -g(xk[1], xk[2], xk[3])
        d[3] = -h(xk[1], xk[2], xk[3])

        # ヤコビ行列の作成
        J[1][1] = f_x(xk[1], xk[2], xk[3])
        J[1][2] = f_y(xk[1], xk[2], xk[3])
        J[1][3] = f_z(xk[1], xk[2], xk[3])
        J[2][1] = g_x(xk[1], xk[2], xk[3])
        J[2][2] = g_y(xk[1], xk[2], xk[3])
        J[2][3] = g_z(xk[1], xk[2], xk[3])
        J[3][1] = h_x(xk[1], xk[2], xk[3])
        J[3][2] = h_y(xk[1], xk[2], xk[3])
        J[3][3] = h_z(xk[1], xk[2], xk[3])
        d = gauss( J, d, N ) # 連立一次方程式を解く
        for i in range(1, N+1):
            xk[i] += d[i]
        k += 1

        if vector_norm1(d) <= EPS or k >= KMAX:
            break

    if k == KMAX:
        print("答えが見つかりませんでした")
    else:
        print(f"答えは x={xk[1]}, y={xk[2]}, z={xk[3]} です")


def f(x: float, y: float, z: float) -> float:
    return -1.0 - x + x*x - y + y*y + x*z


def g(x: float, y: float, z: float) -> float:
    return -4.0 + 3.0*x*x + x*x*x - 2.0*z + 2.0*y*z - z*z + z*z*z


def h(x: float, y: float, z: float) -> float:
    return -2.0*x - 3.0*y + 3.0*x*y - 4.0*z + 2.0*x*z + 4.0*y*z


def f_x(x: float, y: float, z: float) -> float:
    return -1.0 + 2.0*x + z


def f_y(x: float, y: float, z: float) -> float:
    return -1.0 + 2.0*y


def f_z(x: float, y: float, z: float) -> float:
    return x


def g_x(x: float, y: float, z: float) -> float:
    return 6.0*x + 3.0*x*x


def g_y(x: float, y: float, z: float) -> float:
    return 2.0*z


def g_z(x: float, y: float, z: float) -> float:
    return -2.0 + 2.0*y - 2.0*z + 3.0*z*z


def h_x(x: float, y: float, z: float) -> float:
    return -2.0 + 3.0*y + 2.0*z


def h_y(x: float, y: float, z: float) -> float:
    return -3.0 + 3.0*x + 4.0*z


def h_z(x: float, y: float, z: float) -> float:
    return -4.0 + 2.0*x + 4.0*y


if __name__ == "__main__":
    main()
```