```python
import torch
'''一个简单的使用反向传播函数计算梯度的例子'''
def example1():
    # 1、指明需要计算梯度
    x = torch.arange(4.0, requires_grad=True)
    print('x=', x)
    # 2、记录目标值的计算（此处y是标量）
    y = 2 * torch.dot(x, x)
    print('y=', y)
    # 3、执行反向传播函数（backward()函数只能应用于标量）
    y.backward()
    # 4、访问得到的梯度
    print('y关于x的每个分量的梯度为: ', x.grad)

'''对于非标量向量的反向传播'''
def example2():
    x = torch.arange(4.0, requires_grad=True)
    y = x * x                       # y 是一个和 x 形状相同的张量
    # y.sum().backward()            # 对张量 y 进行求和后会降维，使得张量 y 变为标量 y
    y.backward(torch.ones(len(x)))  # 传递一个与 x 具有相同形状的张量，使得每个分量的梯度都为1，即使 y 是一个非标量张量也不会产生问题
    print('y关于x的每个分量的梯度为: ', x.grad)

'''如何进行分离计算: 我们希望只考虑 x 在 y 被计算后发挥的作用'''
def example3():
    x = torch.arange(4.0, requires_grad=True)
    y = x * x
    u = y.detach()          # 将 y 的值赋给 u，且将 u 作为一个常数处理，梯度不会向后流经 u 到 x
    z = u * x
    z.sum().backward()
    print(x.grad == u)

    x.grad.zero_()          # 由于记录了 y 的计算结果，因此我们可以在 y 上调用反向传播函数
    y.sum().backward()
    print(x.grad == 2 * x)
```

```python
'''Python 控制流的梯度计算--即使构建函数的计算需要通过Python控制流（条件、判断、循环等），我们也可以计算得到变量的梯度'''
def f(a):
    b = a * 2
    # 若 b 的 L2 范数小于1000，则一直循环----当 b 的绝对值小于 1000 时，结束循环
    while b.norm() < 1000:
        b = b * 2
    if b.sum() > 0:          # 若 b > 0
        c = b
    else:
        c = 100 * b
    return c
def example4():
    # 定义一个需要计算梯度的随机数标量
    a = torch.randn(size=(), requires_grad=True)
    d = f(a)                 # d = 2a 或 d = 200a
    d.backward()
    print(a.grad == d/a)

if __name__=="__main__":
    example4()
```