task 2

导入基本需要的计算以及绘图模块:

```
1  dec = 6 # 设置每一步计算保留小数点后位数（精度，可以自己调整）
2  import numpy as np
3  np.set_printoptions(formatter={'float': ('{: 0.' + str(dec) + 'f}').format})
4  import matplotlib.pyplot as plt
5  import matplotlib as mpl
6  mpl.rcParams['text.usetex'] = True
7  import sympy as sp
```

1 习题2.11

迭代格式一:

$$x_{k+1} = \ln x_k + 2$$

```
1  def iterationFunc(x):
2      return np.round(np.log(x) + 2, dec)
3      pass
4
5  step = 10
6  xk = np.zeros(step)
7  x0 = 2
8  xk[0] = x0
9  for i in range(step-1):
10     xk[i+1] = iterationFunc(xk[i])
11     pass
12
13 error = xk[1:] - xk[:-1]
14
15 print("xk= ", xk)
16 print("error= ", error)
```
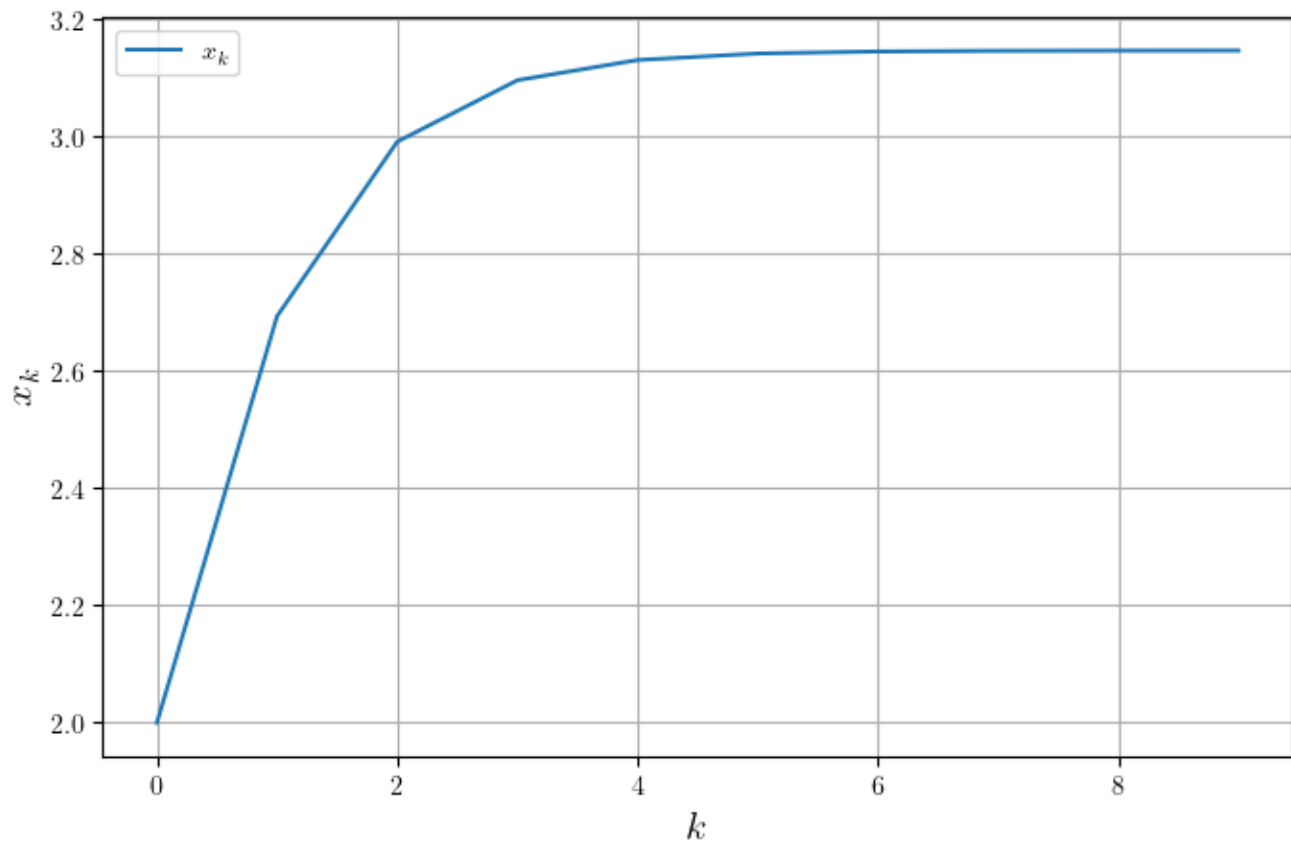
```
1  xk=  [ 2.000000  2.693147  2.990710  3.095511  3.129953  3.141018  3.144547
2    3.145670  3.146027  3.146140]
3  error=  [ 0.693147  0.297563  0.104801  0.034442  0.011065  0.003529  0.001123
4    0.000357  0.000113]
```

```
1  plt.figure(figsize=(8, 5), facecolor="white")
2  plt.plot(np.arange(0, step), xk, label=r"$x_k$")
3  plt.legend()
4  plt.xlabel("$k$", fontsize=15)
5  plt.ylabel("$x_k$", fontsize=15)
6  plt.grid(True)
7  plt.show()
```

调整一下迭代格式，并只计算4步如下：

$$\begin{cases} y_k = \ln x_k + 2 \\ z_k = \ln y_k + 2 \\ x_{k+1} = x_k - \dfrac{(y_k - z_k)^2}{z_k + x_k - 2y_k} \end{cases}$$

```
1   def iterationFuncStephenson(xi):
2       yi = iterationFunc(xi)
3       zi = iterationFunc(yi)
4       x_next = xi - (yi-xi)**2 / (zi+xi-2*yi)
5       return np.around(x_next, dec)
6       pass
7
8   step = 4
9   xk = np.zeros(step)
10  x0 = 2
11  xk[0] = x0
12  for i in range(step-1):
13      xk[i+1] = iterationFuncStephenson(xk[i])
14      pass
15
16  error = xk[1:] - xk[:-1]
17
18  print("xk= ", xk)
19  print("error= ", error)
```
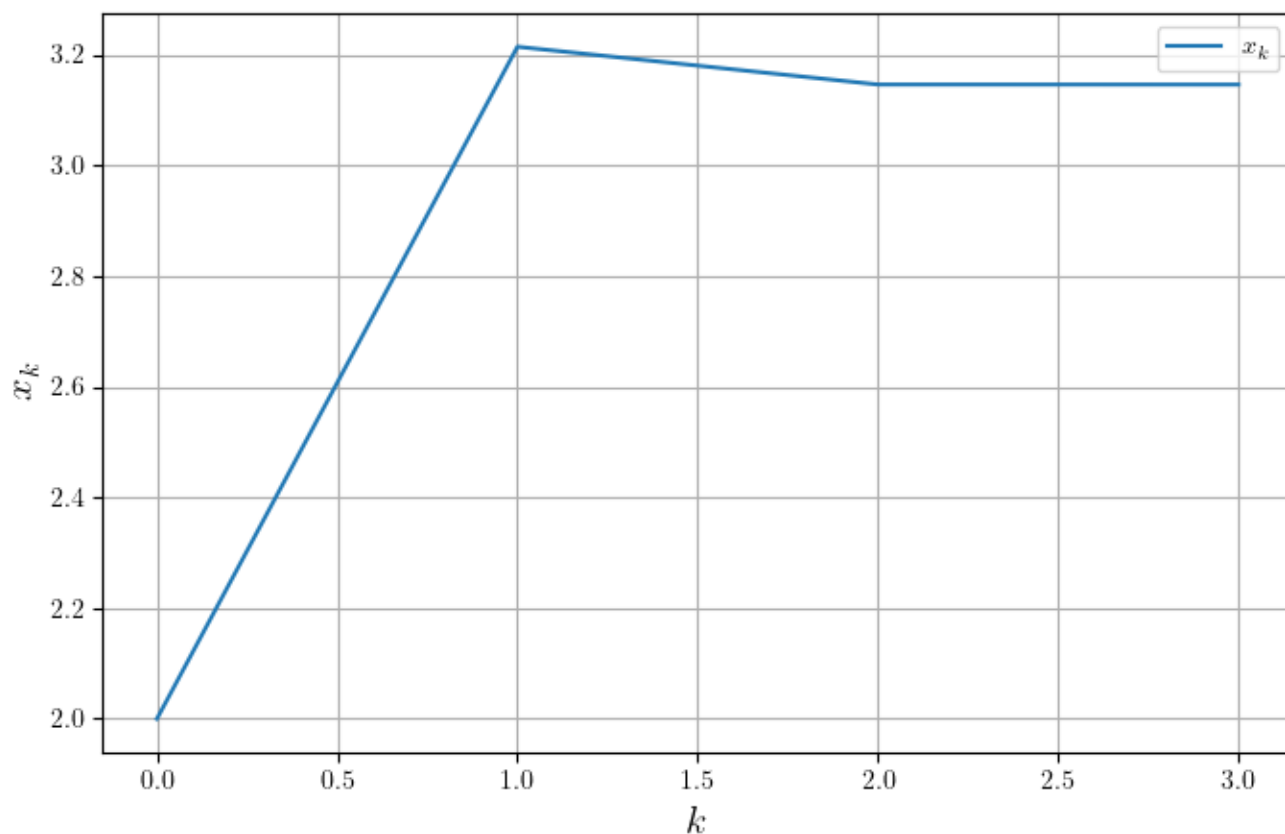
```
1  xk=  [ 2.000000  3.214540  3.146300  3.146193]
2  error=  [ 1.214540 -0.068240 -0.000107]
```

```
1  plt.figure(figsize=(8, 5), facecolor="white")
2  plt.plot(np.arange(0, step), xk, label=r"$x_k$")
3  plt.legend()
4  plt.xlabel("$k$", fontsize=15)
5  plt.ylabel("$x_k$", fontsize=15)
6  plt.grid(True)
7  plt.show()
```


效果拔群。

2 习题2.13

```
1  x_k, x_k_next, x_real, a, p, q, r = sp.symbols('x_k, x_{k+1}, x^*, a, p, q, r')
2  expr = p*x_k + q*a/x_k**2 + r*a**2/x_k**5
3  x_real = a**sp.Rational(1, 3)
4  expr
5  iter = sp.Eq(x_k_next, expr)
6  print("迭代格式: ")
7  iter
```

```
1  迭代格式:
```

$$x_{k+1} = \frac{a^2 r}{x_k^5} + \frac{aq}{x_k^2} + px_k$$

```
1  print("真实解: ")
2  x_real
```

```
1  真实解:
```

$$\sqrt[3]{a}$$

```
1  expr1 = expr.diff(x_k)
2  print("一阶导: ")
3  expr1
```

```
1  一阶导:
```

$$-\frac{5a^2 r}{x_k^6} - \frac{2aq}{x_k^3} + p$$

```
1  expr2 = expr1.diff(x_k)
2  print("二阶导: ")
3  expr2
```

```
1  二阶导:
```

$$\frac{30a^2 r}{x_k^7} + \frac{6aq}{x_k^4}$$

由此建立三个等式如下:

```
1  eq1 = sp.Eq(expr.subs(x_k, x_real), x_real)
2  eq1
```

$$\sqrt[3]{a}p + \sqrt[3]{a}q + \sqrt[3]{a}r = \sqrt[3]{a}$$

```
1  eq2 = sp.Eq(expr1.subs(x_k, x_real), 0)
2  eq2
```

$$p - 2q - 5r = 0$$

```
1  eq3 = sp.Eq(expr2.subs(x_k, x_real), 0)
2  eq3
```

$$\frac{6q}{\sqrt[3]{a}} + \frac{30r}{\sqrt[3]{a}} = 0$$

```
1  res = sp.solve(
2      [eq1, eq2, eq3],
3      [p, q, r]
4  )
5  print("p, q, r: ")
6  res
```

```
1  p, q, r:
```

```
1  {p: 5/9, q: 5/9, r: -1/9}
```

查看一下三阶导数，代入上述的 p, q, r 和 $x = a^{\frac{1}{3}}$：

```
1  expr3 = expr2.diff(x_k).subs(x_k, x_real).subs(res)
2  expr3
```

$$\frac{10}{a^{\frac{2}{3}}}$$

显然上式是不满足等于零的，因此这个迭代格式三阶收敛。

3　习题2.14

显然这个序列满足如下迭代格式:

$$x_{k+1} = \sqrt{x_k + 2}$$

因此稍加计算即可:

```python
def iterationFunc(xi):
    return np.around(
        np.sqrt(xi+2),
        dec
    )
    pass

x0 = 0
step = 20
xk = np.zeros(step)
xk[0] = x0
for i in range(step-1):
    xk[i+1] = iterationFunc(xk[i])
    pass
print("xk=\n", xk)
```

```
xk=
 [ 0.000000  1.414214  1.847759  1.961571  1.990370  1.997591  1.999398
   1.999849  1.999962  1.999990  1.999997  1.999999  2.000000  2.000000
   2.000000  2.000000  2.000000  2.000000  2.000000  2.000000]
```

```python
plt.figure(figsize=(8, 5), facecolor="white")
plt.plot(np.arange(0, step), xk, label=r"$x_k$")
plt.legend()
plt.xlabel("$k$", fontsize=15)
plt.ylabel("$x_k$", fontsize=15)
plt.grid(True)
plt.show()
```
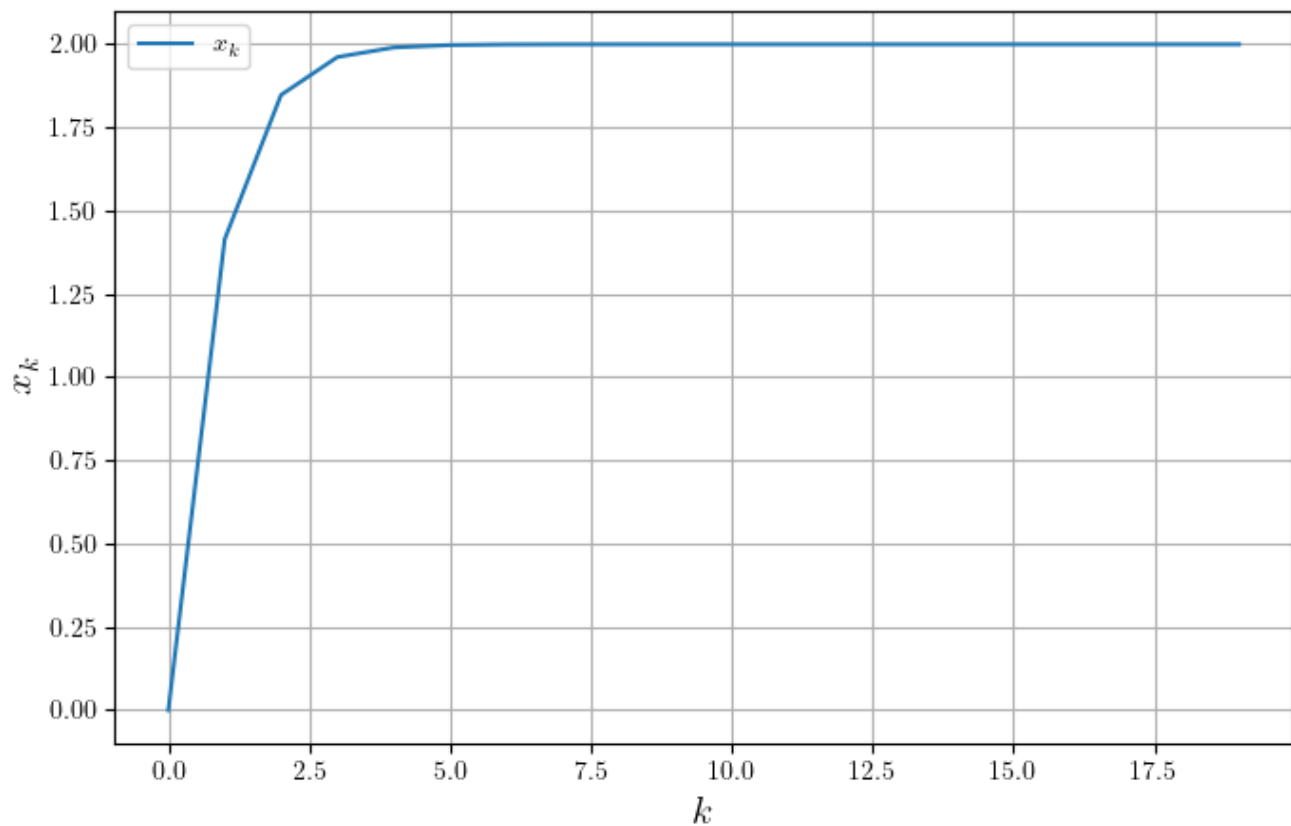
4 习题2.16

事实上这是求解方程的解:

$$f(x) = x^2 - \frac{1}{a} = 0$$

也就是方程:

$$g(x) = a - \frac{1}{x^2} = 0$$

[ref](ref)

```
1  x = sp.symbols("x", positive=True)
2  a = sp.symbols("a", positive=True)
3  expr = a - 1/x**2
4  expr1 = expr.diff(x)
5  print("原函数: ")
6  expr
```

```
1  原函数:
```

$$a - \frac{1}{x^2}$$

```
1   print("一阶导: ")
2   expr1
```

```
1   一阶导:
```

$$\frac{2}{x^3}$$

```
1   expr_iter = (x - expr/expr1).expand()
2   print("迭代格式: ")
3   expr_iter
```

```
1   迭代格式:
```

$$-\frac{ax^3}{2} + \frac{3x}{2}$$

5   习题2.17

```
1   expr = 2*x**2 + 2*x - sp.exp(2*x) + 1
2   expr1 = expr.diff(x)
3   iter_expr = x - expr / expr1
4   iterationFunc = sp.lambdify(x, iter_expr, "numpy")
5   print("表达式: ")
6   expr
```

```
1   表达式:
```

$$2x^2 + 2x - e^{2x} + 1$$

```
1   print("一阶导: ")
2   expr1
```

```
1   一阶导:
```

$$4x - 2e^{2x} + 2$$

```
1  print("迭代格式: ")
2  iter_expr
```

```
1  迭代格式:
```

$$x - \frac{2x^2 + 2x - e^{2x} + 1}{4x - 2e^{2x} + 2}$$

```
1  x0 = 0.5
2  step = 40
3  xk = np.zeros(step)
4  xk[0] = x0
5  for i in range(step-1):
6      xk[i+1] = iterationFunc(xk[i])
7      pass
8  print("xk=\n", xk)
```

```
1  xk=
2   [ 0.500000  0.348053  0.239056  0.162643  0.109929  0.073967  0.049618
3     0.033216  0.022206  0.014831  0.009900  0.006605  0.004406  0.002938
4     0.001959  0.001306  0.000871  0.000581  0.000387  0.000258  0.000172
5     0.000115  0.000076  0.000051  0.000034  0.000023  0.000015  0.000010
6     0.000007  0.000005  0.000002  0.000002  0.000002  0.000002  0.000002
7     0.000002  0.000002  0.000002  0.000002  0.000002]
```

我这里取了40步进行计算，效果如上，一直熟练不到 0 ，接下来我用修正牛顿进行操作：

```
1  m = 3
2  iter_expr = x - expr / expr1 * m
3  iterationFunc = sp.lambdify(x, iter_expr, "numpy")
4  print("修正牛顿迭代公式: ")
5  iter_expr
```

```
1  修正牛顿迭代公式:
```

$$x - \frac{3 \cdot \left(2x^2 + 2x - e^{2x} + 1\right)}{4x - 2e^{2x} + 2}$$

```
x0 = 0.5
step = 40
xkk = np.zeros(step)
xkk[0] = x0
for i in range(step-1):
    xkk[i+1] = iterationFunc(xkk[i])
    pass
print("xkk=\n", xkk)
```

```
xkk=
 [ 0.500000   0.044158   0.000327   0.000000   0.000000   0.000000   0.000000
   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000   0.000000
   0.000000   0.000000   0.000000   0.000000   0.000000]
```
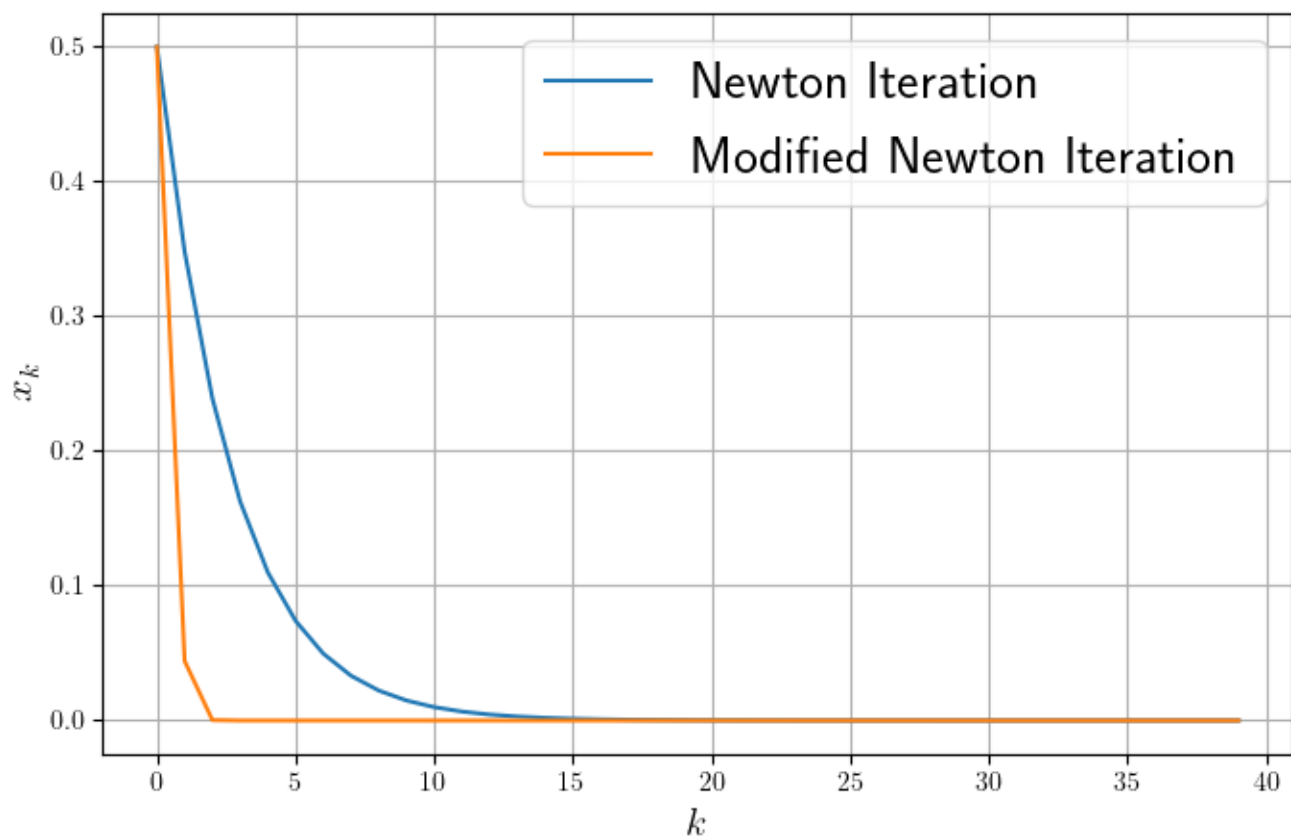
只能说效果拔群↑:

```
plt.figure(figsize=(8, 5), facecolor="white")
plt.plot(np.arange(0, step), xk, label="Newton Iteration")
plt.plot(np.arange(0, step), xkk, label="Modified Newton Iteration")
plt.legend(fontsize=20)
plt.xlabel("$k$", fontsize=15)
plt.ylabel("$x_k$", fontsize=15)
plt.grid(True)
plt.show()
```

6 习题2.19

```
1  expr = (x**3 - 3*x**2 + 3*x - 1) * (x+3)
2  expr = expr.factor()
3  print("原表达式: ")
4  expr
```

```
1  原表达式:
```

$$(x - 1)^3 (x + 3)$$

```
1  print("一阶导函数: ")
2  expr1 = expr.diff(x)
3  expr1 = expr1.factor()
4  expr1
```

```
1  一阶导函数:
```

$$4(x-1)^2(x+2)$$

```
1  m = 3
2  x_k1 = sp.symbols("x_{k+1}")
```

```
1  print("x = 1 局部牛顿迭代收敛表达式: ")
2  sp.Eq(x_k1, x_k - expr.subs(x, x_k) / expr1.subs(x, x_k)*m)
```

```
1  x = 1 局部牛顿迭代收敛表达式:
```

$$x_{k+1} = x_k - \frac{3\left(x_k-1\right)\left(x_k+3\right)}{4\left(x_k+2\right)}$$

```
1  print("x = -3 局部牛顿迭代收敛表达式: ")
2  sp.Eq(x_k1, x_k - expr.subs(x, x_k) / expr1.subs(x, x_k))
```

```
1  x = -3 局部牛顿迭代收敛表达式:
```

$$x_{k+1} = x_k - \frac{\left(x_k-1\right)\left(x_k+3\right)}{4\left(x_k+2\right)}$$