求解非线性方程 $3x^2 - e^x = 0$

图形

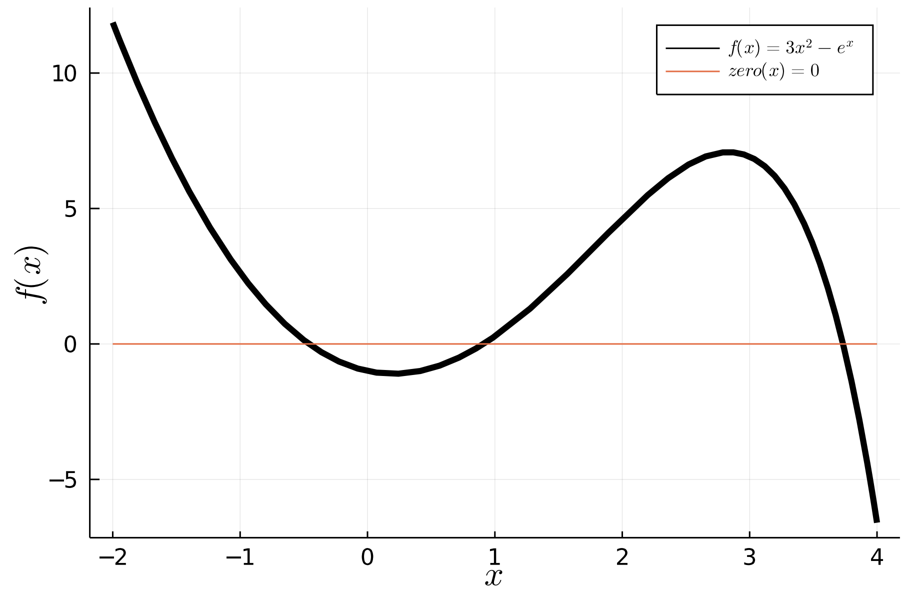

图 1: 函数$f(x) = 3x^2 - e^x$在$x \in (-2, 4)$中的图像。由图可见在该区间内有三个零点，大概位于$X_0 \approx -0.5, 0.9, 3.8$

牛顿(Newton)迭代法

考虑一般的一元非线性方程$f(x) : \mathbb{R} \to \mathbb{R}$，目标是找到一个$x$，且满足：

$$f(x) = 0 \tag{1}$$

任意选定一个x_0作为初始的猜测解。一般地，$f(x_0) \neq 0$. 因而需要找到Δx_0 使得$f(x_0 + \Delta x_0) = 0$。对$f(x_0 + \Delta x_0)$做泰勒展开：

$$f(x_0 + \Delta x_0) = f(x_0) + \frac{df(x)}{dx}\Big|_{x_0} \Delta x_0 + \mathcal{O}(\Delta x_0^2) \tag{2}$$

这里只保留到一阶小量。我们希望该次迭代就是方程的解：

$$f(x_1) \equiv f(x_0 + \Delta x_0) = 0 \iff \Delta x_0 \approx -f(x_0) \Big/ \left(\frac{df(x)}{dx}\right)\Big|_{x_0} \tag{3}$$

上式定义了一次迭代$x_1 = x_0 + \Delta x_0$。如果本次迭代满足$f(x_1) = 0$，则x_1就是方程的解。否则，以相同的方法进行下一次迭代：$f(x_1 + \Delta x_1) = 0$。对于第n次迭代：

$$x_{n+1} = x_n - f(x_n) \Big/ \left(\frac{df(x)}{dx} \right) \Big|_{x_n} \tag{4}$$

以此类推，直到精度达到要求。

收敛分析：

假设$x = X$是方程$f(x) = 0$的解。另外假设对于某$n > N$, $|X - x_n| = \delta \ll 1$，即x_n是一个接近解的估计值。不失一般地，假定$x_n < X$。现做二阶泰勒展开：

$$f(X) = f(x_n + \delta) = f(x_n) + \frac{df(x)}{dx}\Big|_{x_n} (X - x_n) + \frac{1}{2} \frac{d^2 f(x)}{dx^2}\Big|_{\alpha} (X - x_n)^2 = 0 \tag{5}$$

这里假设了$f(x)$连续，且在二阶中使用了中值定理，用$f''(\alpha)$代替$\mathcal{O}(\delta^3)$，其中$\alpha \in [X, x_n]$。另外由式(4)可得：

$$f(x_n) = \frac{f(x)}{dx}\Big|_{x_n} (x_n - x_{n+1}) \tag{6}$$

由式(5, 6)可得：

$$f'(x_n)(X - x_{n+1}) + \frac{f''(\alpha)}{2} (X - x_n)^2 = 0 \tag{7}$$

现定义$e_n = X - x_n$为第n次迭代的误差，则上式可写作：

$$e_{n+1} = -\frac{f''(\alpha)}{2f'(x_n)} e_n^2 \tag{8}$$

即误差收敛速度为二次方误差e_n^2。

数值计算：

我们希望得到$X \in [3, 4]$，因而根据图1，可以选择初始猜测值为$x_0 = 3.5$。计算使用Julia实现。牛顿迭代法的函数代码如下：

```
function newton(f::Function, fprime::Function, paralist::Tuple)

    x0, tolerance, epsilon, maxIterations, solutionFound = paralist
    histIteration = Float64[x0]    # Iteration history

    for i = 1:maxIterations        # Begin iteration
      y = f(x0)
      yprime = fprime(x0)

      if abs(yprime) < epsilon     # Stop if the denominator is too small
        break
      end

      global x1 = x0 - y/yprime    # Do Newton’s computation
```

```
      push!(histIteration, x1)     # Push x1 to history array

      # Stop when the result is within the desired tolerance
      if abs(x1 - x0) <= tolerance
        solutionFound = true
        break
      end

      x0 = x1                 # Update x0 to start the process again
    end

    if solutionFound
      println("Solution: ", x1)
      println("\nError:", f(x1))
    else
      println("Did not converge")
    end

    return x1, f(x1), histIteration

end
```

计算结果：

如图，牛顿迭代法在六次迭代后收敛。计算结果为$x = 3.733$, 误差$\sim 7e - 15$

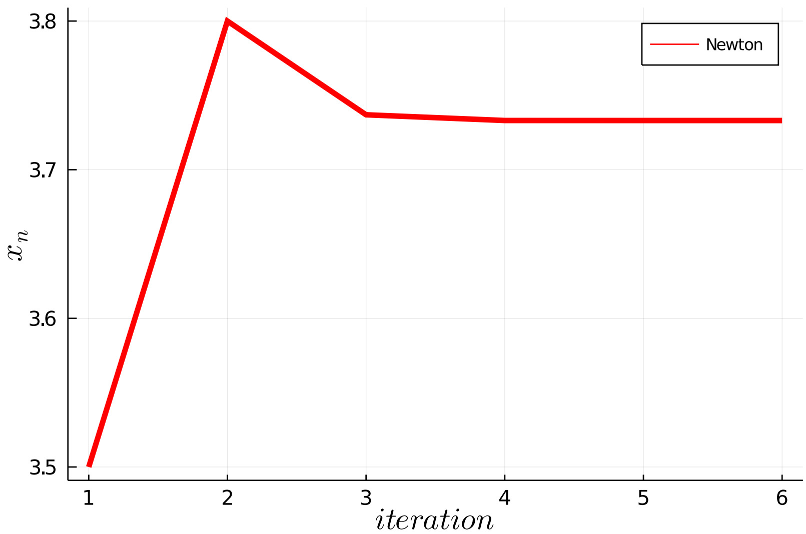

图 2: 牛顿迭代法的收敛过程。精度为$1e - 9$

斯特芬森(Steffensen)迭代法

与牛顿迭代法相似。唯一的区别在于斯特芬森不需要调用函数的导函数$f'(x)$。迭代过程为

$$x_{n+1} = x_n - \frac{f(x_n)}{g(x_n)} \tag{9}$$

其中$g(x)$定义为：

$$g(x) = \frac{f[x + f(x)] - f(x)}{f(x)} = \frac{f[x + f(x)]}{f(x)} - 1 \tag{10}$$

与牛顿迭代法相比，它的一个优点在于迭代过程不需要求得函数的导数。对于复杂的函数更加方便。

收敛分析：

与牛顿迭代法的分析同理。同样是以二次方速度收敛: $e_{n+1} \propto e_n^2$

数值计算：

斯特芬森实现的Julia函数代码如下：

```
function steffensen(f::Function, paralist::Tuple)

    x0, tolerance, epsilon, maxIterations, solutionFound = paralist
    histIteration = Float64[x0]    # Iteration history

    for i = 1:maxIterations        # Begin iterations
      y = f(x0)
      g = f(x0 + f(x0)) / f(x0) - 1

      if abs(g) < epsilon          # Stop if the denominator is too small
        break
      end

      global x1 = x0 - y/g         # Do Steffensen's computation
      push!(histIteration, x1)     # Push x1 to history array

      # Stop when the result is within the desired tolerance
      if abs(x1 - x0) <= tolerance
        solutionFound = true
        break
      end

      x0 = x1              # Update x0 to start the process again
    end

    if solutionFound
      println("Solution: ", x1)
      println("\nError:", f(x1))
    else
```

```
        println("Did not converge")
    end

    return x1, f(x1), histIteration
end
```

计算结果：

如图，斯特芬森迭代法在18次迭代后收敛。计算结果与牛顿法相同：$x = 3.733$, 误差$\sim 7e-15$

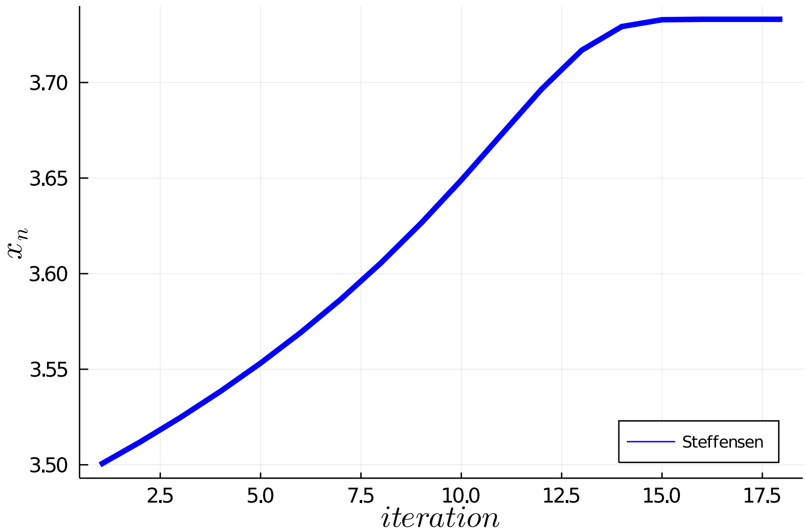

图 3: 斯特芬森迭代法的收敛过程。精度为$1e-9$

数值实验所用参数

在两种迭代的计算中所使用的相关参数如下：

```
f(x) = 3x^2 - exp(x)             # The function to solve
fprime(x) = 6x - exp(x)          # The derivative of the function

# Parameters for Newton iterations
x0 = 3.5                         # The initial guess
tolerance = 1e-9                 # 9 digit accuracy
maxIterations = 20               # Maximum iterations
solutionFound = false            # Flag
```