Solution of Nonlinear Equations

## 1. Introduction

* 방정식의 해를 구하는 것은 통계학에서 빈번하게 발생하는 문제이다.
* 비선형(nonlinear) 방정식은 closed-form solution이 존재하지 않는 경우가 많다.
* 수치적으로 방정식의 해를 구할(근사할) 수 있는데, 이는 적절한 반복알고리즘에 의해 얻어진다.
* 3가지 방법(Bisection, Functional iteration, Newton's method)에 대해서 학습한다.
* 각 방법의 장단점을 파악하고 실제 사례에 응용하여 본다.

## 2. Bisection method (이분법)

* 연속함수의 해가 특정 폐구간 안에 존재할 때, 해가 존재하는 구간을 반씩 좁혀 해를 구하는 방법.
* 방정식의 해를 구하기 위한 가장 간단한 방법 중 하나이다.
* 장점
  + 최소한의 조건만 만족하면 항상 방정식의 해를 얻을 수 있음.
  + 함수의 미분이 필요없음.
* 단점
  + 다른 방법들에 비해 상대적으로 계산속도가 느림.

Algorithm : 해 근사

* 가 에서 연속이고 일 때,
  + step 1 : 를 계산
  + step 2 : 이면, c가 방정식의 해. 그렇지 않으면, , 를 계산. 면, 를 로, 면, 를 로 대체하여 새로운 구간을 생성함.
  + step 3 : 새로운 구간에 대해 step1-2를 반복함. 구간의 폭이 미리 정한 수준보다 작아지면 멈추고, 해당 구간의 중점이 방정식의 근사해가 됨.
* 알고리즘이 한 번 시행될 때마다 구간의 폭은 씩 줄어든다.
* 중간값정리(intermediate value theorem)에 의해 알고리즘의 수렴이 보장된다.
* Example
* 의 해를 구하여라. (폐구간 [3,4])

## 3. Functional Iteration (함수반복법)

* 방정식의 해를 구하는 문제를 고정점(fixed point)을 구하는 문제로 바꾸어 푸는 방법.

cf) 고정점 : f(x)=x 를 만족하는 x.

* 장점
  + 알고리즘이 매우 직관적이고 간단함.
  + 함수의 미분이 필요없음.
* 단점
  + 함수의 변화율이 제어되지 않으면 수렴성이 보장되지 않음.

Algorithm : 의 고정점 근사

* step 1 : 초기값 설정.
* step 2 : 를 에 대해서 반복하여 계산하여 수열 을 생성.
* step 3 : 수열의 변화가 미리 정한 수준 이하로 떨어지면 멈추고 그 때의 수열값이 고정점의 근사값.
* 초기값과 함수의 성질에 따라 알고리즘이 수렴할 수도, 발산할 수도 있다.
* 특정 조건을 만족하면, 아래와 같은 정리에 따라 수렴성이 보장된다.

정리

함수 가 폐구간 에서 정의된 함수일 때,

1. 이면, 항상 이다.
2. 임의의 에 대하여 를 만족하는 상수 가 존재한다.

조건 a,b가 만족되면, 함수 는 구간 에서 유일한 고정점 을 가지고, 위 알고리즘은 초기값에 상관없이 항상 로 수렴한다.

cf) 이 때 를 Lipschitz constant라고 한다.

* Example
* 의 고정점을 구하여라. (초기값 1, 1/16)

## 4. Newton's method (뉴튼방법)

* 이 의 해 의 좋은 근사치라고 가정하면, 평균값정리(mean value theorem)로부터

를 만족시키는 이 과 사이에 존재한다. 를 , 를 으로 각각 대체하면 다음 식을 얻는다.

* 1차 테일러근사(taylor approximation)에 의해 근사치를 업데이트해 나가는 것으로 이해할 수도 있다. 을 현재 근사치라 하면, 근방에서 는 다음과 같이 근사된다.

풀고자 하는 방정식은 이므로 이 대신 근사함수를 이용하여 을 풀면,

이 해가 되고, 이 값이 새로운 근사치 이 된다.

* 장점
  + 알고리즘의 수렴속도가 매우 빠르다.
* 단점
  + 알고리즘의 수렴이 초기값의 선택에 매우 강하게 의존한다.
  + 함수의 미분이 필요하다.

Algorithm : 의 해 근사

* step 1 : 초기값 설정.
* step 2 : 를 에 대해서 반복하여 계산하여 수열 을 생성.
* step 3 : 수열의 변화가 미리 정한 수준 이하로 떨어지면 멈추고 그 때의 수열값이 고정점의 근사값.
* 초기값을 방정식의 해에 매우 가깝게 잡으면 수렴성이 보장된다.
* Example
* 의 해를 구하여라. (초기값 5, 2.5, 2, )

## 5. Examples

#### 5.1 Computation of Quantiles

* 분포함수가 인 연속형확률분포의 -분위수(quantile)를 계산하고자 하는 문제는 다음과 같은 방정식을 푸는 문제로 생각할 수 있다.

표준정규분포의 95% 분위수를 찾아보자.

이분법

* 먼저 해를 포함하고 있는 폐구간을 찾는다. 으로 설정하였다.
* 이므로, 이 성립한다.

a=0;b=3  
g <- function(x){pnorm(x)-0.95}  
sign(g(0)\*g(3))

## [1] -1

tol <- 10^{-8}  
for (i in 1:100)  
 {  
 c=(a+b)/2  
 newa=c\*(g(c)\*g(b)<0)+a\*(1-(g(c)\*g(b)<0))  
 newb=c\*(g(c)\*g(a)<0)+b\*(1-(g(c)\*g(a)<0))  
 a <- newa ; b <- newb  
 err <- abs(a-b)  
 print(c(i,a,b))  
 if (err<tol) break  
 }

## [1] 1.0 1.5 3.0  
## [1] 2.00 1.50 2.25  
## [1] 3.000 1.500 1.875  
## [1] 4.0000 1.5000 1.6875  
## [1] 5.00000 1.59375 1.68750  
## [1] 6.000000 1.640625 1.687500  
## [1] 7.000000 1.640625 1.664062  
## [1] 8.000000 1.640625 1.652344  
## [1] 9.000000 1.640625 1.646484  
## [1] 10.000000 1.643555 1.646484  
## [1] 11.000000 1.643555 1.645020  
## [1] 12.000000 1.644287 1.645020  
## [1] 13.000000 1.644653 1.645020  
## [1] 14.000000 1.644836 1.645020  
## [1] 15.000000 1.644836 1.644928  
## [1] 16.000000 1.644836 1.644882  
## [1] 17.000000 1.644836 1.644859  
## [1] 18.000000 1.644848 1.644859  
## [1] 19.000000 1.644854 1.644859  
## [1] 20.000000 1.644854 1.644856  
## [1] 21.000000 1.644854 1.644855  
## [1] 22.000000 1.644854 1.644854  
## [1] 23.000000 1.644854 1.644854  
## [1] 24.000000 1.644854 1.644854  
## [1] 25.000000 1.644854 1.644854  
## [1] 26.000000 1.644854 1.644854  
## [1] 27.000000 1.644854 1.644854  
## [1] 28.000000 1.644854 1.644854  
## [1] 29.000000 1.644854 1.644854

(a+b)/2

## [1] 1.644854

g((a+b)/2)

## [1] 1.329473e-10

함수반복법

* 초기값을 1.5으로 설정하였다.

x=1.5  
g <- function(x){pnorm(x)-0.95 + x}  
tol <- 10^{-8}  
for (i in 1:100)  
 {  
 newx <- g(x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 x<-newx  
 }

## [1] 1.0 1.5  
## [1] 2.000000 1.483193  
## [1] 3.000000 1.464181  
## [1] 4.000000 1.442609  
## [1] 5.000000 1.418044  
## [1] 6.000000 1.389955  
## [1] 7.000000 1.357683  
## [1] 8.000000 1.320401  
## [1] 9.000000 1.277051  
## [1] 10.000000 1.226258  
## [1] 11.000000 1.166208  
## [1] 12.000000 1.094443  
## [1] 13.000000 1.007562  
## [1] 14.0000000 0.9007297  
## [1] 15.0000000 0.7668637  
## [1] 16.0000000 0.5952824  
## [1] 17.0000000 0.3694551  
## [1] 18.00000000 0.06356083  
## [1] 19.0000000 -0.3610991  
## [1] 20.0000000 -0.9520865  
## [1] 21.00000 -1.73156  
## [1] 22.000000 -2.639884  
## [1] 23.000000 -3.585737  
## [1] 24.000000 -4.535569  
## [1] 25.000000 -5.485566  
## [1] 26.000000 -6.435566  
## [1] 27.000000 -7.385566  
## [1] 28.000000 -8.335566  
## [1] 29.000000 -9.285566  
## [1] 30.00000 -10.23557  
## [1] 31.00000 -11.18557  
## [1] 32.00000 -12.13557  
## [1] 33.00000 -13.08557  
## [1] 34.00000 -14.03557  
## [1] 35.00000 -14.98557  
## [1] 36.00000 -15.93557  
## [1] 37.00000 -16.88557  
## [1] 38.00000 -17.83557  
## [1] 39.00000 -18.78557  
## [1] 40.00000 -19.73557  
## [1] 41.00000 -20.68557  
## [1] 42.00000 -21.63557  
## [1] 43.00000 -22.58557  
## [1] 44.00000 -23.53557  
## [1] 45.00000 -24.48557  
## [1] 46.00000 -25.43557  
## [1] 47.00000 -26.38557  
## [1] 48.00000 -27.33557  
## [1] 49.00000 -28.28557  
## [1] 50.00000 -29.23557  
## [1] 51.00000 -30.18557  
## [1] 52.00000 -31.13557  
## [1] 53.00000 -32.08557  
## [1] 54.00000 -33.03557  
## [1] 55.00000 -33.98557  
## [1] 56.00000 -34.93557  
## [1] 57.00000 -35.88557  
## [1] 58.00000 -36.83557  
## [1] 59.00000 -37.78557  
## [1] 60.00000 -38.73557  
## [1] 61.00000 -39.68557  
## [1] 62.00000 -40.63557  
## [1] 63.00000 -41.58557  
## [1] 64.00000 -42.53557  
## [1] 65.00000 -43.48557  
## [1] 66.00000 -44.43557  
## [1] 67.00000 -45.38557  
## [1] 68.00000 -46.33557  
## [1] 69.00000 -47.28557  
## [1] 70.00000 -48.23557  
## [1] 71.00000 -49.18557  
## [1] 72.00000 -50.13557  
## [1] 73.00000 -51.08557  
## [1] 74.00000 -52.03557  
## [1] 75.00000 -52.98557  
## [1] 76.00000 -53.93557  
## [1] 77.00000 -54.88557  
## [1] 78.00000 -55.83557  
## [1] 79.00000 -56.78557  
## [1] 80.00000 -57.73557  
## [1] 81.00000 -58.68557  
## [1] 82.00000 -59.63557  
## [1] 83.00000 -60.58557  
## [1] 84.00000 -61.53557  
## [1] 85.00000 -62.48557  
## [1] 86.00000 -63.43557  
## [1] 87.00000 -64.38557  
## [1] 88.00000 -65.33557  
## [1] 89.00000 -66.28557  
## [1] 90.00000 -67.23557  
## [1] 91.00000 -68.18557  
## [1] 92.00000 -69.13557  
## [1] 93.00000 -70.08557  
## [1] 94.00000 -71.03557  
## [1] 95.00000 -71.98557  
## [1] 96.00000 -72.93557  
## [1] 97.00000 -73.88557  
## [1] 98.00000 -74.83557  
## [1] 99.00000 -75.78557  
## [1] 100.00000 -76.73557

x

## [1] -77.68557

g(x)

## [1] -78.63557

* 초기값을 바꿔가며 시험해 보아라. 수렴하지 않는다면 이유는 무엇인가?

뉴튼방법

* 초기값을 1.5로 설정하였다.

x=1.5  
g <- function(x){pnorm(x)-0.95}  
dg<- function(x){dnorm(x)}  
tol <- 10^{-8}  
for (i in 1:100)  
 {  
 newx <- x - g(x)/dg(x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 x<-newx  
 }

## [1] 1.0 1.5  
## [1] 2.000000 1.629768  
## [1] 3.000000 1.644669  
## [1] 4.000000 1.644854  
## [1] 5.000000 1.644854

x

## [1] 1.644854

g(x)

## [1] -1.110223e-16

* 초기값을 0으로 설정하였다.

x=0  
g <- function(x){pnorm(x)-0.95}  
dg<- function(x){dnorm(x)}  
tol <- 10^{-8}  
for (i in 1:100)  
 {  
 newx <- x - g(x)/dg(x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 x<-newx  
 }

## [1] 1 0  
## [1] 2.000000 1.127983  
## [1] 3.000000 1.505239  
## [1] 4.000000 1.630773  
## [1] 5.000000 1.644693  
## [1] 6.000000 1.644854  
## [1] 7.000000 1.644854

x

## [1] 1.644854

g(x)

## [1] 0

* 초기값을 3으로 설정하였다.

x=3  
g <- function(x){pnorm(x)-0.95}  
dg<- function(x){dnorm(x)}  
tol <- 10^{-8}  
for (i in 1:100)  
 {  
 newx <- x - g(x)/dg(x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 if (abs(dg(x))<10^{-30}) break  
 x<-newx  
 }

## [1] 1 3  
## [1] 2.000000 -7.977384  
## [1] 3.000000e+00 1.569539e+14

x

## [1] 1.569539e+14

g(x)

## [1] 0.05

* 초기값의 설정에 따라 알고리즘이 어떻게 작동하는지 살펴보고, 왜 그러한지 생각해 보아라.

#### 5.2 Extinction Probabilities (branching process)

* 현재 개체가 개의 자손으로 번성할 확률을 라 하자. 1개의 개체로부터 이러한 과정이 계속되었을 때, 결국에는 소멸할 확률을 라 할 때 이 확률을 계산하여 보자.
* 번째 세대에서 소멸할 확률을 이라 하면, 이 성립한다. 즉, 라 하면, 이다.
* 는 를 만족해야 한다. 즉, 의 고정점이다. (Why?)
* 의 성질
  + 이 성립. ()
  + 라고 가정해도 좋다. (Why?)
  + , . 즉, 는 단조증가하는 볼록(convex)함수이다.
  + 이면, 즉, 무조건 소멸하지만 이면, 로 소멸이 불확실. 여기서 은 특정개체의 평균자손을 나타낸다. (Why?)
* 미국의 어떤 지역에서 특정 성씨에 대한 자손생성함수 가 다음과 같이 주어졌을 때, Extinction probablity를 구하여 보아라.
* by functional iteration

x=0  
g <- function(s)(0.4982+0.2103\*s+0.1270\*s^2+0.0730\*s^3+0.0418\*s^4+0.0241\*s^5+0.0132\*s^6+0.0069\*s^7+0.0035\*s^8+0.0015\*s^9+0.0005\*s^10)  
tol <- 10^{-4}  
for (i in 1:100)  
 {  
 newx <- g(x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 x<-newx  
 }

## [1] 1 0  
## [1] 2.0000 0.4982  
## [1] 3.0000000 0.6471058  
## [1] 4.0000000 0.7187532  
## [1] 5.0000000 0.7606954  
## [1] 6.0000000 0.7880596  
## [1] 7.0000000 0.8071817  
## [1] 8.0000000 0.8211892  
## [1] 9.0000000 0.8318062  
## [1] 10.0000000 0.8400622  
## [1] 11.0000000 0.8466109  
## [1] 12.0000000 0.8518872  
## [1] 13.000000 0.856192  
## [1] 14.0000000 0.8597403  
## [1] 15.0000000 0.8626897  
## [1] 16.0000000 0.8651583  
## [1] 17.0000000 0.8672368  
## [1] 18.0000000 0.8689952  
## [1] 19.0000000 0.8704891  
## [1] 20.0000000 0.8717627  
## [1] 21.0000000 0.8728518  
## [1] 22.0000000 0.8737854  
## [1] 23.0000000 0.8745876  
## [1] 24.000000 0.875278  
## [1] 25.0000000 0.8758733  
## [1] 26.0000000 0.8763873  
## [1] 27.0000000 0.8768316  
## [1] 28.0000000 0.8772161  
## [1] 29.0000000 0.8775491  
## [1] 30.0000000 0.8778377  
## [1] 31.000000 0.878088  
## [1] 32.0000000 0.8783053  
## [1] 33.000000 0.878494  
## [1] 34.0000000 0.8786578  
## [1] 35.0000000 0.8788002  
## [1] 36.0000000 0.8789241  
## [1] 37.0000000 0.8790317

x

## [1] 0.8790317

g(x)

## [1] 0.8791254

* by Newton's method

x=0  
g <- expression(0.4982+0.2103\*s+0.1270\*s^2+0.0730\*s^3+0.0418\*s^4+0.0241\*s^5+0.0132\*s^6+0.0069\*s^7+0.0035\*s^8+0.0015\*s^9+0.0005\*s^10 - s)  
dg <- D(g,"s")  
tol <- 10^{-4}  
for (i in 1:100)  
 {  
 newx <- x - eval(g,list(s=x))/eval(dg,list(s=x))  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 if ((abs(eval(dg,list(s=x))))<10^{-30}) break  
 x<-newx  
 }

## [1] 1 0  
## [1] 2.0000000 0.6308725  
## [1] 3.0000000 0.7996846  
## [1] 4.0000000 0.8599923  
## [1] 5.0000000 0.8775963  
## [1] 6.0000000 0.8797221

x

## [1] 0.8797221

#### 5.3 Division without dividing

* 의 역수는 의 해로 정의된다. 본 방정식을 뉴튼의 방법
* 에 의해 반복근사하여 구할 수 있다.

a=0.2  
x=2  
tol <- 10^{-8}  
for (i in 1:100)  
 {  
 newx <- x\*(2-a\*x)  
 err <- abs(newx-x)  
 print(c(i,x))  
 if (err<tol) break  
 x<-newx  
 }

## [1] 1 2  
## [1] 2.0 3.2  
## [1] 3.000 4.352  
## [1] 4.000000 4.916019  
## [1] 5.000000 4.998589  
## [1] 6 5  
## [1] 7 5

x

## [1] 5