hwk4.R

evanjohnston

Sun Feb 14 15:25:27 2016

```r
# Evan Johnston
# M348 - hwk 4 - feb. 16 2016

# Newton's Method

# This method uses derivatives, so we require a package to find
#  accurate numerical derivatives
require('numDeriv')
```

```
## Loading required package: numDeriv
```

```
## Warning: package 'numDeriv' was built under R version 3.1.3
```

```r
# inputs:
#   p0: initial approximation
#   tol: tolerance level of error
#   n: max number of iterations
#   func: function to iterate over
newton<-function(p0, tol, n, func){

  # matrix to format output
  final<-matrix(rep(0,n*2), nrow=n)
  colnames(final)<-c('iteration', 'Root')

  # initialize counter
  k<-1

  # while loop to iterate up to n times
  while (k<=n){

    # find p(k) for current iteration
    p<-p0-func(p0)/grad(func, p0)

    # if found or error within tolerance then output and exit
    if (abs(p-p0)<tol){

      # vector of outputs
      output<-c(k,p)
      final[k,]<-output
      return (final[1:k,])
    }

    # record current output
    output<-c(k,p)
    final[k,]<-output
```

```
    # update p0
    p0<-p

    # iterate counter
    k<-k+1
  }

  # record last result
  output<-c(k,p)
  final[k,]<-output
  return (final)
}

# create input function to be iterated over
f1<-function(x){
  return (exp(1)^x+2^(-x)+2*cos(x)-6)
}

# run the Newton's Method function with:
#   function f1=e^x+2^(-x)+2*cos(x)-6
#   inital guess 1.5
#   with error limit 10e-6 for 1000 iterations
result1<-newton(1.5,10e-6,1000,f1)
print(result1)
```

```
##      iteration     Root
## [1,]         1 1.956490
## [2,]         2 1.841533
## [3,]         3 1.829506
## [4,]         4 1.829384
## [5,]         5 1.829384
```

```
############################################################

# Secant Method
# inputs:
#   p0: initial approximation
#   p1: second approximation
#   tol: tolerance level of error
#   n: max number of iterations
#   func: function to iterate over
secant<-function(p0, p1, tol, n, func){

  # matrix to format output
  final<-matrix(rep(0,n*2), nrow=n)
  colnames(final)<-c('iteration', 'Root')

  # initialize counter at 2
  k<-2

  # find output of initial guesses
  q0<-func(p0)
  q1<-func(p1)
```

```
  # while loop to iterate up to n times
  while (k<=n){

    # find p(k) for current iteration
    p<-p1 - (q1*(p1 - p0))/(q1-q0)

    # if found or error within tolerance then output and exit
    if (abs(p-p1)<tol){

      # vector of outputs
      output<-c(k,p)
      final[k,]<-output
      return (final[1:k,])
    }

    # record current output
    output<-c(k,p)
    final[k,]<-output

    # update p0, p1, q0, q1
    p0<-p1
    p1<-p
    q0<-q1
    q1<-func(p)

    # iterate counter
    k<-k+1
  }

  # record last result
  output<-c(k,p)
  final[k,]<-output
  return (final)
}

# create input function to be iterated over
f2<-function(x){
  return (x^2-4*x+4-log(x))
}

# run the Secant Method function with:
#   function f2=x^2-4x-ln(x)+4
#   inital guess 3, 3.1
#   with error limit 10e-7 for 1000 iterations
result2<-secant(3, 3.1, 10e-7,1000,f2)
print(result2)
```

```
##      iteration     Root
## [1,]         0 0.000000
## [2,]         2 3.055647
## [3,]         3 3.057068
## [4,]         4 3.057104
## [5,]         5 3.057104
```