# Dekker Method

Dekker’s method is root finding algorithm combining the bisection method and the secant method. The algorithm tries to use the potentially fast converging secant method with the reliability of bisection method.

Dekker gave this in 1969

Initial input:

We want to solve the equation

Assuming that is continuous in and and have opposite signs.

Three points are used at every step for finding the root

1. represents the current iterate and this is equal to current root of the
2. is other bound of the interval such that and have opposite signs and there is guaranteed a root in the interval .

Assumption: and the is a better guess than .

1. represents the previous iteration and for initial iteration

Iterative formula:

The second solution is due to contribution of bisection method thus

Finding the :

Finding the new contra point:

Finally putting constraint on and

If this means that is the better guess thus we will swap the and

We will continue this until we get

|  |
| --- |
| from math import sin  from math import cos  from math import e  def w(x):      y =(4\*sin(100\*x) + 3\*cos(150\*x) + 5\*sin(170\*x)+ 8\*cos(190\*x) +0.09)/50      return abs(y)  #function for which we have to find the root  def f(x):      # y = 7\*e\*\*(-x)\*sin(x)-1      y = e\*\*(mu\*x - (sigma\*sigma\*x/2) + sigma\*w(x)) - k      return y  #dekker method  def dekker(a,b,fun\_tol,maxit):      ak = a      bk = b      fak = f(ak)      fbk = f(bk)      iteration = 0;        #checking whether the intial bounds are not the solution      if abs(fbk) <= fun\_tol:          root = bk          return bk,iteration        elif abs(fak) <= fun\_tol:          root = ak          return ak,iteration      #assumption      if abs(fak) < abs(fbk):          ak,bk = bk,ak          fak,fbk = fbk,fak          #function has no root in this interval      if fak\*fbk >0:            print("Stock Price Can't be doubled ")              #the existence of root in such an interval is not guaranteed          return 0,iteration      #multiple iteration      bk\_1 = ak      while abs(bk - bk\_1) > fun\_tol and iteration <maxit:          fbk\_1 = f(bk\_1);          m = (ak + bk)/2          if fbk\_1 != fbk: #secant method              xr = bk - fbk\*(bk - bk\_1)/(fbk - fbk\_1)              if bk<m:                  if xr<m and xr>bk:                      bk1 = xr                  else:                      bk1 = m              elif bk>m:                  if xr>m and xr<bk:                      bk1 = xr                  else:                      bk1 = m          else: #bisection method              xr = (ak + bk)/2              bk1 = m          iteration= iteration + 1          fbk1 = f(bk1)          fxr = f(xr)            if abs(fxr) <= fun\_tol:              return xr,iteration          if fak\*fxr < 0:              ak1 = ak          else:              ak1 = bk          if abs(f(ak1)) < abs(f(bk1)):              ak,bk = bk,ak              fak,fbk = fbk,fak          bk\_1 = bk;          bk = bk1;          ak = ak1;          fak = f(ak);          fbk = f(bk1);          if iteration >= maxit:              return xr,iteration  #input ---------------------  mu = .006  sigma = 0.000012  k = 2  a=1  b = 4000  fun\_tol = 1e-12;  maxit = 1000  #-----------------------------------  root,it\_count = dekker(a,b,fun\_tol,maxit)  print(root)  print(it\_count)  # ---------------------------------------- |