**Algorithm 1: Bisection method**

|  |
| --- |
| Method introduction: |
| The method is applicable for numerically solving the equation f(x) = 0 for the real variable x, where f is a continuous function defined on an interval [a, b] and where f(a) and f(b) have opposite signs. In this case a and b are said to bracket a root since, by the intermediate value theorem, the continuous function f must have at least one root in the interval (a, b).  At each step the method divides the interval in two by computing the midpoint c = (a+b) / 2 of the interval and the value of the function f(c) at that point. Unless c is itself a root (which is very unlikely, but possible) there are now only two possibilities: either f(a) and f(c) have opposite signs and bracket a root, or f(c) and f(b) have opposite signs and bracket a root.[5] The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step. In this way an interval that contains a zero of f is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.  Explicitly, if f(a) and f(c) have opposite signs, then the method sets c as the new value for b, and if f(b) and f(c) have opposite signs then the method sets c as the new a. (If f(c)=0 then c may be taken as the solution and the process stops.) In both cases, the new f(a) and f(b) have opposite signs, so the method is applicable to this smaller interval. |
| Algorithm Design |
| Step1:Calculate *c*, the midpoint of the interval,  Step2:Calculate the function value at the midpoint, *f*(*c*).  Step3:If convergence is satisfactory (that is, *c* - *a* is sufficiently small, or |*f*(*c*)| is sufficiently small), return *c* and stop iterating.  Step3：Examine the sign of *f*(*c*) and replace either (*a*, *f*(*a*)) or (*b*, *f*(*b*)) with (*c*, *f*(*c*)) so that there is a zero crossing within the new interval. |
| Matlab code |
| function [x, y] = MyBisect(fun, a0, b0, tol, max)  % This is the code for Bisection method.  % Input:  % [a0, b0] Initial interval  % fun function  % tol Allowable tolerance in computed zero  % max Maximum number of iterations  % Output:  % x Vector of approximations to zero  % y Vector of function values, fun(x)  % Preallocate vectors.  x = zeros(max, 1);  y = zeros(max, 1);  a = zeros(max, 1);  b = zeros(max, 1);  ya = zeros(max, 1);  yb = zeros(max, 1);  % Set an intial interval.  a(1) = a0; b(1) = b0;  ya(1) = feval(fun, a(1)); yb(1) = feval(fun, b(1));  % Check whether the intial interval is a bracket.  if ya(1)\*yb(1) > 0  error('Function has same sign at end points\n');  end  % Bisection search  for i = 1 : max  x(i) = (a(i) + b(i))/2;  y(i) = feval(fun, x(i));  if (x(i) - a(i) < tol)  fprintf('Bisection method has converged\n');  break;  end  if y(i) == 0  fprintf('Exact solution found\n');  break;  elseif y(i)\*ya(i) < 0  a(i+1) = a(i); ya(i+1) = ya(i);  b(i+1) = x(i); yb(i+1) = y(i);  else  a(i+1) = x(i); ya(i+1) = y(i);  b(i+1) = b(i); yb(i+1) = yb(i);  end  iter = i+1;  end  if (iter > max)  fprintf('Zero not found to desired tolerance within the maximum number of iterations\n');  end  % Output results  k = 1:iter;  fprintf(' iter a b x y\n');  disp([k' a(1:iter) b(1:iter) x(1:iter) y(1:iter)]);  % This is the driver file for MyBisect.  % Input:  % [a, b] Initial interval  % fun function  % tol Allowable tolerance in computed zero  % max Maximum number of iterations  % Output:  % x Approximated solution x  % y fun(x)  clear;  close all;  clc  a = 1;  b = 2;  tol = 1e-2;  max = 20;  fun = 'cos';  [x, y] = MyBisect(fun, a, b, tol, max); |
| Examples and Result |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Iteration** | **{\displaystyle a\_{n}}** | **{\displaystyle b\_{n}}** | **{\displaystyle c\_{n}}** | **{\displaystyle f(c\_{n})}** | | 1 | 1 | 2 | 1.5 | −0.125 | | 2 | 1.5 | 2 | 1.75 | 1.6093750 | | 3 | 1.5 | 1.75 | 1.625 | 0.6660156 | | 4 | 1.5 | 1.625 | 1.5625 | 0.2521973 | | 5 | 1.5 | 1.5625 | 1.5312500 | 0.0591125 |   Remarks |
|  |
|  |