

## Fixed-Point Project

A fixed point is a solution of the equation  $x = g(x)$ . Fixed point iteration can be used to find roots of equations. This method is not described in your text. Consider the problem of solving

$$5x^3 - 10x + 3 = 0.$$

Rewriting as

$$10x = 5x^3 + 3,$$

we can reformulate the problem as finding a fixed point of the equation

$$x = g(x) = 0.5x^3 + 0.3.$$

The iteration formula for this example is

$$x_{n+1} = 0.5x_n + 0.3.$$

Taking  $x_0 = 0.1$  as a starting estimate and rounding the result at each step to five digits, we find that

$$x_1 = 0.5(0.1^3) + 0.3 = .3005,$$

$$x_2 = .5(0.3005^3) + 0.3 = .31357,$$

$$x_3 = .5(0.31357^3) + 0.3 = .31542.$$

Eventually, such a sequence of iterations may converge. The theorem in the notes describe when that is likely to happen.

Use the algorithm Fixed-Point Iteration on page 22 of the class notes to write a procedure fixedpoint that implements the algorithm. For parameters, you will need the function g (type algebraic), the first approximation p0 (type numeric), a tolerance tol (type positive), a maximum number of iterations no (type posint), and a return variable root (type name). You can model your work on the workshhet newtonalg.mws, which is my implementation of the algorithm Newton's, for the procedure newton in our class library. The coding for that procedure is below. Test the procedure with the above example and a tolerance of  $10^{-5}$ . Also test it with the formulation of the same problem that we get by adding to each side of the equation,

$$x = g(x) = 5x^3 - 9x + 3.$$

You do not need to put in the step comments found below.

```

> newton := proc(f::algebraic,p0::complex,tol::positive,no::posint,
root::name) local F, FP, OK, P0, TOL, NO, F0, I, FP0, D, COMP;
P0:=evalf(p0);
NO:=no;
TOL:=evalf(tol);
FP := unapply(diff(f,x),x);
F := unapply(f,x);
printf(`  i      p          f(p)\n`);
printf(`  -      -          ----\n`);
F0 := evalf(F(P0));
STEP 1
  I := 0;
  printf(`%3d    %-30a    %-30a\n`,I,P0,F0);
  OK := TRUE;
STEP 2
  while I <= NO and OK = TRUE do
STEP 3
  compute P(I)
    FP0 := evalf(FP(P0));
    D := F0/FP0;
STEP 6
    P0 := P0 - D;
    F0 := evalf(F(P0));
    printf(`%3d    %-30a    %-30a\n`,I+1,P0,F0);
STEP 4
    if abs(D) < TOL then
procedure completed sucessfully
      printf(`\nThe approximate solution is %a`,args[5]);
      printf(` = %-a \n`,P0);
      printf(`with f(%a`,args[5]);
      printf(`) = %-a\n`,F0);  OK := FALSE;
STEP 5
    else
      I := I+1;
    end if;
  end do;
  if OK = TRUE then
STEP 7
  procedure completed unsuccessfully
    printf(`\nIteration number %3d`,NO);
    printf(` gave approximation %-a\n`,P0);
    printf(`F(P) = %-a not within tolerance : %-a\n`,F0,TOL);
    RETURN();
  else
    P0;
  end if;
end proc;

```

Warning, imaginary unit `I` used as a local variable in procedure newton

```

newton := proc( f::algebraic, p0::complex, tol::positive, no::posint, root::name )
    local F, FP, OK, P0, TOL, NO, F0, I, FP0, D, COMP;
    P0 := evalf( p0 );
    NO := no;
    TOL := evalf( tol );
    FP := unapply( diff( f, x ), x );
    F := unapply( f, x );
    printf( ` i p          f(p)
          - - -          ----
          `); F0 := evalf( F( P0 ) ); I := 0; printf( `%3d %-30a %-30a
`, I, P0, F0);
    OK := TRUE;
    while I <= NO and OK = TRUE do
        FP0 := evalf( FP( P0 ) );
        D := F0 / FP0;
        P0 := P0 - D;
        F0 := evalf( F( P0 ) );
        printf( `%3d %-30a %-30a
          `, I + 1, P0, F0); if abs(D) < TOL then printf( `
The approximate solution is %a`, args[5]); printf( ` = %-a
`, P0); printf( `with f(%a`, args[5]); printf( `) = %-a
`, F0); OK := FALSE else I := I + 1 end if end do; if OK = TRUE then printf( `
Iteration number %3d`, NO); printf( ` gave approximation %-a
`, P0); printf( ` F(P) = %-a not within tolerance : %-a
`, F0, TOL); RETURN( ) else P0 end if end proc

```