**fsolve** - solve using floating-point arithmetic

**Calling Sequence:**

`fsolve( eqns, vars, options );`

**Parameters:**

- `eqns` - an equation or set of equations, or a procedure
- `vars` - (optional) an unknown or set of unknowns
- `options` - (optional) parameters controlling solutions

**Description:**

- The conventions for passing equations and variables, and returning the answers, are the same for `fsolve` as for `solve`.

- For a general equation, `fsolve` attempts to compute a single real root. However, for polynomials of one variable it will compute all real (non-complex) roots, although exceptionally ill-conditioned polynomials may cause `fsolve` to miss some roots.

- To compute all roots of a polynomial over the field of complex numbers, use the `complex` option.

- The options available are:
  
  * `avoid=s` --- Avoid certain values when searching for roots. `s` is a set of equations of the form `var = value`.
  
  * `complex` --- Find one root (or all roots, for polynomials) over the complex floating-point numbers.
  
  * `fulldigits` --- This option prevents `fsolve` from decreasing `Digits` for intermediate calculations at high settings of `Digits`. With this option `fsolve` may escape ill-conditioning problems, but the routine slows down.
  
  * `maxsols=n` --- Find only the `n` least roots. This option is only meaningful for polynomials, where more than one root is computed.
  
  * `interval` --- Specifically: `a..b` or `x = a..b` or `{x=a..b, y=c..d, ...}`. Search for roots in the given interval only. The ranges are closed intervals, i.e. the endpoints are included in the range.

- You can specify starting values for the variables to solve for by giving an equation of the form `var = value` instead of just the variables to solve for in `vars`. If the method that `fsolve` chooses for a particular problem requires more than one starting value, as many values as are required are generated by perturbing the given starting values.

- Note that an `fsolve` computation may fail to find a root even though one exists, in which case specifying appropriate range information may result in a successful computation.
In the real case, if `fsolve` returns unevaluated it is because no starting point converged to a solution. In this case, specifying a range may be very helpful.

**Examples:**

```plaintext
> fsolve( tan(sin(x))=1, x );

   .9033391108

> poly := 23*x^5 + 105*x^4 - 10*x^2 + 17*x:
   fsolve( poly, x, -1..1 );

   -.6371813185 0.

> fsolve( poly, x, maxsols=3 );

   -4.536168981 -.6371813185 0.

> q := 3*x^4 - 16*x^3 - 3*x^2 + 13*x + 16:
   fsolve(q, x, 1..2);

   1.324717957

> fsolve(q, x, 2..5);

> fsolve(q, x, 4..8);

   5.333333333

> fsolve(q, x, complex);

   -0.6623589786 .5622795121 + -0.6623589786 .5622795121 I 1.324717957 5.333333333

> f := sin(x+y) - exp(x)*y = 0:
   g := x^2 - y = 2:
   fsolve({f,g},{x,y},{x=-1..1,y=-2..0});

   { x = -.6687012050, y = -1.552838698 }

> f:=10-(ln(v+(v^2-1)^(1/2))-ln(3+(3^2-1)^(1/2)))

   f := 10 - \ln(\sqrt{v^2 - 1} + 3) + \frac{1}{2} \ln(3 + \sqrt{8} )

> fsolve(f,v);

   64189.82535

> fsolve(f,v,1..infinity);
```
> fsolve(sin(x), x, avoid={x=0, x=Pi, x=-Pi}, -10..10);

-6.283185307

> fsolve(sin(x), x=3.1);

3.141592654

> f := proc(x) x-cos(x) end proc:
fsolve(f, 1.0);

.7390851332

See Also:
solve, dsolve, isolve, msolve, rsolve