[6.10] Exact solutions to cubic and quartic equations

The 89/92+ functions *csolve()* and *czeros()* will not always return exact solutions to cubic and quartic equations. The routines *cubic()* and *quartic()* can be used to get the exact solutions.

cubic():

```
cubic(é,ý)
Func
©˜cubic(p,v) Roots of cubic equ
 p - cubic equ
 v - independent var

Local  p,r,q,ù,a,b,u,v,w
If  getType(é)≠"EXPR":Return  "p\cubic:arg must be a cubic eq"
p\coef(é,ý)→ù
If  dim(ù)≠4:Return  "p\cubic:arg must be a cubic eq"
string(cZeros(é,ý))→w
If  inString(w,".")=Ø and w≠"{}":Return  expr(w)
ù[2]/(ù[1])→p
ù[3]/(ù[1])→q
ù[4]/(ù[1])→r
(3*q-p^2)/3→a
(2*p^3-9*p*q+27*r)/27→b
(√(3*(4*a^3+27*b^2))-9*b)^(1/3)*2^(2/3)/(2*3^(2/3))→w
If getType(w)="NUM" Then
 If w=Ø Then
  If p=Ø Then
   -1→u
   1→v
  Else
   Ø→u
   Ø→v
  EndIf
 Else
  ω-a/(3*ω)→u
  ω+a/(3*ω)→v
 EndIf
Else
  ω-a/(3*ω)→u
  ω+a/(3*ω)→v
EndIf
{α-p/3,-α/2+√(3)/2*i*β-p/3,-α/2-√(3)/2*i*β-p/3,α=u,β=v,ω=w}
©Copyright 1998,1999 Glenn E. Fisher
EndFunc
```

quartic():

```
quartic(pp,ý)
Func
©˜quartic(p,v) Roots of 4th degree
© polynomial
© p - the polynomial
© v - variable

Local qq,ù,ú,r,d,e,y,j,k
If getType(ý)≠"VAR": Return "p\quartic:2nd arg must be a VAR"
p\coef(pp,ý)→ù
If dim(ù)≠5: Return "p\quartic:1st arg must be a 4th degree"
cZeros(pp,ý)→r
If inString(string(r),".")=Ø: Return r
ù/(ù[1])→ù
ý^3-ù[3]*ý^2+(ù[4]*ù[2]-4*ù[5])*ý+4*ù[3]*ù[5]-ù[4]^2-ù[2]^2*ù[5]→qq
```

```
p\cubic(qq,ý)→ú
p\evalrt(ú)→ú
Ø→k
For j,1,dim(ú)
 If real(ú[j])=ú[j] Then
  j→k
  Exit
 EndIf
EndFor
ú[k]→y
√(ù[2]^2/4-ù[3]+y)→r
If r=Ø Then
 √(3*ù[2]^2/4-2*ù[3]+2*√(y^2-4*ù[5]))→d
 √(3*ù[2]^2/4-2*ù[3]-2*√(y^2-4*ù[5]))→e
Else
 √(3*ù[2]^2/4-r^2-2*ù[3]+(4*ù[2]*ù[3]-8*ù[4]-ù[2]^3)/(4*r))→d
 √(3*ù[2]^2/4-r^2-2*ù[3]-(4*ù[2]*ù[3]-8*ù[4]-ù[2]^3)/(4*r))→e
EndIf
{-ù[2]/4+r/2+d/2,-ù[2]/4+r/2-d/2,-ù[2]/4-r/2+e/2,-ù[2]/4-r/2-e/2}
EndFunc
```

coef():

```
coef(è,ý)
Func
©˜coef(p,v) Make list of coefficents
 p - polynomial
 v - independent var

Local  ù,é,ì
If  getType(ý)≠"VAR":Return  "p\coef:2nd arg must be a VAR"
If inString(string(è),"=")>Ø Then
 left(è)→é
Else
 è→é
EndIf
é|ý=Ø→ù[1]
1→ì
Loop
 d(é,ý)→é
 If  getType(é)="NUM" and é=Ø:Exit
 ì+1→ì
 augment({(é|ý=Ø)/((ì-1)!)},ù)→ù
EndLoop
ù
EndFunc
```

evalrt():

```
evalrt(l)
Func
©˜evalRt(ls) Evaluate Roots
 ls - list of roots

Local  i,n,o,s,f
If  getType(l)≠"LIST": Return  "p\evalRt:Arg must be a LIST"
dim(l)→n
{}→o
Ø→f
For  i,1,n
 If inString(string(l[i]),"=")>Ø Then
  If f=Ø Then
   l[i]→s
```

```
    1→f
   Else
    s and l[i]→s
   EndIf
  Else
   augment(o,{l[i]})→o
  EndIf
 EndFor
 If f=0 Then
  o
 Else
  o|s
 EndIf
 EndFunc
```

cubic() is used to solve cubic (third-order) equations, and *quartic()* solves 4th-order equations. *coef()* and *evalrt()* are used by *cubic()* and *quartic()* to find the solutions. *evalrt()* can also be used on the results from *cubic()* and *quartic()*, as described below, to expand the solutions.

All of these routines must be in the same folder, called *p*. The mode must be set to Auto, and *not* to Exact or Approx. Set Complex format to Rectangular or Polar. To help prevent "Memory" errors, these routines should be the only variables in the folder, and there should be no other variables in the folder. These routines can take a long time to return the result.

The input to both *cubic()* and *quartic()* is a polynomial in *x* or *z*. The output is a list of the solutions. For example,

`cubic(x^3-12x^2-15x+26,x)`	returns	{-2 1 13}
`quartic(z^4-2z^3-13z^2+38z-24,z)`	returns	{-4 1 2 3}

The advantage of using *cubic()* and *quartic()* is that they return exact answers, which *czeros()* or *csolve()* don't always do. For example,

```
czeros(x^3-x+1,x)
```

returns

{-1.32472 0.662359 - .56228i 0.662359+.56228i}

while

```
cubic(x^3-x+1,x)
```

returns a list with these solution elements:

$$\alpha$$

$$\frac{-\alpha}{2}+\frac{\beta\cdot\sqrt{3}}{2}\cdot i$$

$$\frac{-\alpha}{2}-\frac{\beta\cdot\sqrt{3}}{2}\cdot i$$

$$\alpha = \omega + \frac{1}{3\cdot\omega}$$

$$\beta = \omega - \frac{1}{3\cdot\omega}$$

$$\omega = \frac{(-3\cdot(\sqrt{69}-9))^{\frac{1}{3}}\cdot 2^{\frac{2}{3}}}{12} + \frac{(9-\sqrt{69})^{\frac{1}{3}}\cdot 3^{\frac{5}{6}}\cdot 2^{\frac{2}{3}}}{12}\cdot i$$

The first three elements are the solutions to the cubic polynomial. The last three elements define the variable substitutions used in the solutions. This format reduces the size of the output list, and makes the solutions more intelligible. The function *evalrt()* automatically applies these substitutions. *evalrt()* is called with the solution list output from *cubic()*, with these results returned as a list:

$$\frac{3^{\frac{2}{3}}\cdot 2^{\frac{1}{3}}}{6\cdot(9-\sqrt{69})^{\frac{1}{3}}} + \frac{(-3\cdot(\sqrt{69}-9))^{\frac{1}{3}}\cdot 2^{\frac{2}{3}}}{12} + \left(\frac{(9-\sqrt{69})^{\frac{1}{3}}\cdot 3^{\frac{5}{6}}\cdot 2^{\frac{2}{3}}}{12} - \frac{3^{\frac{1}{6}}\cdot 2^{\frac{1}{3}}}{2\cdot(9-\sqrt{69})^{\frac{1}{3}}}\right)\cdot i$$

$$\frac{-\left((9-\sqrt{69})^{\frac{2}{3}}\cdot 2^{\frac{1}{3}} + 2\cdot 3^{\frac{1}{3}}\right)\cdot 6^{\frac{1}{3}}}{6\cdot(9-\sqrt{69})^{\frac{1}{3}}}$$

$$\frac{\left((9-\sqrt{69})^{\frac{2}{3}}\cdot 2^{\frac{1}{3}} + 2\cdot 3^{\frac{1}{3}}\right)\cdot 6^{\frac{1}{3}}}{12\cdot(6-\sqrt{69})^{\frac{1}{3}}} - \frac{\left((-3\cdot(\sqrt{69}-9))^{\frac{2}{3}}\cdot 2^{\frac{1}{3}} - 6\right)\cdot 3^{\frac{1}{6}}\cdot 2^{\frac{1}{3}}}{12\cdot(9-\sqrt{69})^{\frac{1}{3}}}\cdot i$$

These solutions can be verified by substituting them back into the original polynomial.

For another example, try quartic(x^4-x+1,x). The results are quite long, and not shown here. Since they involve inverse trigonometric functions, they cannot be checked by substituting them into the original polynomial. However, if they are substituted into the original polynomial and approximate results are found, those results are zero to within machine precision.

If the polynomial to be solved has complex coefficients, you must specify that the argument variable is complex with the underscore character "_", or the complex solutions will not be returned. For example, solve the polynomial generated by

$$(x+2)\cdot(x-3)\cdot(x-7i)$$

which obviously has roots of -2, 3 and 7i. This expression expands to

$$x(x-3)(x+2) - 7(x-3)(x+2)i$$

If you call cubic like this:

```
cubic(x*(x-3)*(x+2)-7*(x-3)*(x+2)*i,x)
```

then the returned solutions are {-2,3}: the complex solution of 7i is not returned. However, if *cubic()* is properly called with the underscore, to indicate that *x* is complex, like this

```
cubic(x_*(x_-3)*(x_+2)-7*(x_-3)*(x_+2)*i,x_)
```

then the correct solutions of {-2, 3, 7i} are returned.

cubic() and *quartic()* first try to find an exact solution with *czeros().* If an exact solution is not returned, they use Cardano's formula to find the solutions. One reference which describes Cardano's formula is the *CRC Standard Mathematical Tables and Formula*, 30th edition, Daniel Zwillinger, Editor-in-Chief.

(Contributor declines credit. Credit also to Glenn Fisher and Pini Fabrizio.)