# ROOTS — Of Polynomials, That Is

*Laguerre's method does not perform as promised*

MATLAB 3.5 has two different methods for finding the roots of polynomials. Both of them have interesting histories, and interesting numerical properties. But we now believe one of them is seriously flawed. We are recommending that it NOT be used. I'd like to explain how and why we came to this conclusion. But first some background.

Let's generate two simple test polynomials, $p(x)$ and $q(x)$, with the MATLAB statements

```
p = poly([1 2 3])
```

and

```
q = poly([2 2 2]).
```

We get

```
p =
        1     -6     11     -6
q =
        1     -6     12     -8
```

The elements in p are the coefficients in the cubic

$$p(x) = x^3 - 6x^2 + 11x - 6$$

The equation $p(x) = 0$ has roots at $x = 1$, $x = 2$, and $x = 3$. The elements in q are the coefficients in the cubic

$$q(x) = x^3 - 6x^2 + 12x - 8$$

The equation $q(x) = 0$ has a triple root at $x = 2$.

What happens to the roots when we perturb the polynomial? Let's focus on the root $x = 2$ and subtract a small quantity, $\varepsilon$, from the constant term in the polynomial. The perturbed equation

$$p(x) = (x - 1)(x - 2)(x - 3) = \varepsilon$$

can be written

$$x = 2 + \varepsilon / ((x - 1)(x - 3))$$

Near $x = 2$ the denominator on the right is nearly equal to -1, so the resulting perturbation in the root is close to $-\varepsilon$; it is certainly the same size as $\varepsilon$.

On the other hand, the perturbed equation

$$q(x) = (x - 2)^3 = \varepsilon$$

has to be written

$$x = 2 + \varepsilon^{1/3}$$

The size of the perturbation in $x = 2$ is the cube root of $\varepsilon$. If $\varepsilon$ is MATLAB's IEEE arithmetic roundoff error, $2^{-52}$, then $x = 2$ is changed to $x = 2.00000605545445$, which hardly appears to be a single rounding error.

More than this happens. When we perturb $p(x)$ by a small amount, its roots remain real. But almost any perturbation of $q(x)$ will cause two of its roots to move into the complex plane.

This is a general phenomenon. If a polynomial has a root of multiplicity $m$, then most perturbations of size $\varepsilon$ will cause the root to be perturbed by amounts of size $\varepsilon^{1/m}$. And, if $m$ is odd, complex roots are almost certain to appear. Moreover, this has nothing to do with any particular root finding method. It is a property of the roots themselves, not of some algorithm.

What does all this have to do with matrices? The connection is through the *companion* matrix. If $p$ is a vector with $n+1$ elements representing the coefficients in a polynomial of degree $n$, then the matrix

```
C = [-p(2:n+1)/p(1); eye(n-1,n)]
```

is the companion matrix associated with p. For example, with

```
p =     [1     -6   11     -6],

C =     [6     -11    6
          1      0    0
          0      1    0]
```

The characteristic polynomial of the companion matrix, $det(\lambda I - C)$, is simply $p(\lambda)$. The eigenvalues of $C$ are the roots of p. Perturbations of size $\varepsilon$ in the elements of $C$ cause an eigenvalue of multiplicity $m$ to be perturbed by the $m$-th root of $\varepsilon$.

Which brings us to MATLAB's original polynomial root finder, `roots(p)`. This is a short M-file that simply sets up the companion matrix and uses the built-in `eig` function to find its eigenvalues. We started using a matrix eigenvalue routine to compute polynomial roots with "classic" MATLAB 12 years ago. We did so for three reasons:

(1) The code is short; we get polynomial roots almost for free using code we already have.

(2) It emphasizes the power of modern algorithms for matrix computation. Before the 1970s, polynomial root finders were used to compute matrix eigenvalues. Now, we've turned the tables.

(3) The algorithm is reasonably accurate and robust.

But this approach is not the best possible. For a polynomial of degree $n$, it uses order $n^2$ storage and order $n^3$ time. An algorithm designed specifically for polynomial roots might use order $n$ storage and $n^2$ time. And the roundoff errors introduced correspond to perturbations in the elements of the companion matrix, not the polynomial coefficients. We don't know of any cases where the computed roots are not the exact roots of a slightly perturbed polynomial, but such cases might exist. †

For our sample polynomial, *q(x)*, with a triple root at $x = 2$, `roots(q)` computes

```
2.00000957563204 + 0.00001658568659i
2.00000957563204 - 0.00001658568659i
1.99998084873593
```

You can see the errors of size $\varepsilon^{1/3}$ in both the real and imaginary parts. These errors correspond to roundoff errors in the original polynomial coefficients, but, still, we would somehow prefer to see three exact 2s.

So, a second polynomial root finder, `roots1`, was introduced into MATLAB several years ago. It uses Laguerre's method and is based on code in the popular book *Numerical Recipes* by W. H. Press et al. Our *MATLAB User's Guide* says "`roots1` performs the same function as `roots`, but gives more accurate answers when there are repeated `roots`." Sure enough, `roots1(q)` gives three exact 2s.

That's not the end of the story. At some point, a line of code was dropped from `roots1`. The statement

```
b = bsave;
```

may be missing just after the comment `Polish each root`. So we have looked more carefully at `roots1`. Even with the missing line restored, it does not always deliver on its promise of more accurate roots.

Multiplying all the coefficients of a polynomial by the same scale factor should not change its roots. But it does lead to different roundoff errors, and so to different values for the computed roots. We find,

```
roots1(q/3) =
        1.99999994838086
        2.00000005161914
        2.00000000000000
```

and

```
roots1(q/5) =
         2.00000000000000 + 0.00000000120521i
         2.00000000000000 - 0.00000004094164i
         2.00000000000000 + 0.00000003973643i
```

The errors in the computed results are comparable in size to those produced by `roots`, and are consistent with the perturbation theory for a multiple root. But there is a disturbing lack of symmetry. The `roots` function finds points on a small circle in the complex plane centered at the exact answer; `roots1` does not.

It gets worse. Here is another example.

```
p = [20  -181  596  -906  596  -181  20]
```

This represents a polynomial of degree 6 whose roots are

```
x = [4 2+i 2-i 1/4 1/(2+i) 1/(2-i)]
```

There are no multiple roots. The roots are not badly conditioned and `roots(p)` computes them to full accuracy. But the polynomial does have the special property that its roots come in reciprocal pairs, $x_k$ and $1/x_k$. This somehow causes `roots1`, with or without the missing line replaced, to produce completely unacceptable results. We will skip the gory details. We haven't pursued the matter far enough to understand exactly why `roots1` fails. If anybody wants to investigate further, or to see if the *Numerical Recipes* code also fails, we would sure like to hear about it. We suspect there will not be an easy fix.

Any roundoff error in the evaluation of a polynomial must lead to perturbations in its computed roots that are roughly the same size as those produced by our eigenvalue-based `roots` function. If, like our example *q(x)*, the polynomial happens to have integer coefficients, and an integer-valued multiple root, then a routine like `roots1` may be able to avoid roundoff error altogether and compute the multiple root exactly. But this is a very special situation, and is insufficient justification to include the routine in MATLAB.

Oh, by the way, `roots1` is also very slow. So, we recommend that you remove it, and `laguer`, from your MATLAB toolbox. We intend to remove them from MATLAB 4.0.

† This note was written in 1991. Since then Alan Edelman and Lloyd Tretethen have written papers showing that the `roots` function is numerically OK– the computed eigenvalues of the companion matrix are always the exact roots of a slightly perturbed polynomial.

*This is a general phenomenon. If a polynomial has a root of multiplicity* m, *then most perturbations of size* $\varepsilon$ *will cause the root to be perturbed by amounts of size* $\varepsilon^{1/m}$.