Rooting for accuracy

Suppose we have a quadratic polynomial $p(x;s) = a(s)x^2 + b(s)x + c(s)$ whose coefficients depend on a parameter s. In this lab, we define

$$p(x;s) = 9x^2 - (6+s)x + 1. \tag{1}$$

For each value of s there are two roots $t_1(s)$ and $t_2(s)$. Since a, b, and c vary continuously as a function of s, $t_1(s)$ and $t_2(s)$ are also continuous.

The effect of roundoff in floating point calculations is to perturb the coefficients of p by a relative amount ϵ_{mach}. As a result, the roots may change by a relative amount $\kappa\epsilon_{\text{mach}}$, where κ is the condition number of the roots. In the text it's shown in a special case that for the root t_k, the condition number is given by

$$\kappa = |t_k|/|t_1 - t_2|. \tag{2}$$

If $t_1 \approx t_2$, the errors in the computed roots are potentially large (compared to machine precision). In our case $p(x;0) = (3x-1)^2$, which makes κ formally infinite, so $s = 0$ is critical to the conditioning of the roots.

Goals

You will examine the sensitivity of the roots of a quadratic polynomial whose roots come together, coalesce, and separate again.

Preparation

Read section 1.2 and the online help for the `roots` and `polyval` commands.

In MATLAB, perform the following.

```
t1 = @(s) (6+s+sqrt((6+s).^2-36))/18;
t2 = @(s) (6+s-sqrt((6+s).^2-36))/18;
s = linspace(-1,1,800)';
plot(t1(s),'r.')
hold on
plot(t2(s),'b.')
xlabel('real part'),  ylabel('imaginary part')
title('Roots for -1 < s < 1')
```

Use the resulting picture to describe in words what happens to the roots as the parameter s passes from negative to positive values.

Procedure

Download the template script and complete it in sections as guided below.

1. Define the functions `t1` and `t2` as in the Preparation section above.

2. Let $s = -0.1$. Use `roots` to compute the two roots of $p(x;s)$ as a vector `r`. One complication is that we don't know whether `r(1)` is meant to be t_1 or t_2. So we compute its error by means of

   ```
   err1 = min( abs(r(1)-t1(s)), abs(r(1)-t2(s)) )
   ```

   Find this value. Then find κ from (2) with $k = 1$.

3. Define the vector

   ```
   s = linspace(-1,1,800)';
   ```

   In a loop over the entries of the vector `s`, find the error and κ, storing the results in vectors. The template has code for plotting the results.

4. Nothing in the previous step indicates much reason for concern. However, the results are strikingly different if we approach very close to $s = 0$. Define

   ```
   s = logspace(-15,-1,100)';
   ```

   Then repeat the loop from step 3.

Going further

We always compute with a fixed value of ϵ_{mach}, but we can simulate the effect of varying it by manually perturbing data. For example, a relative perturbation of size δ to each entry of vector y is made by

```
r = 2*rand(size(y))-1;    % random in [-1,1]
yy = y .* (1+delta*r);
```

For each $\delta = 10^{-6}, 10^{-8}, 10^{-10}, 10^{-12}, 10^{-14}$, make 500 random perturbations to the coefficient vector of $p(x;0)$, each time using `roots` to find the resulting roots and plotting them in the complex plane. (You should end up five plots, each with 1000 points.)