Math 466/666: Programming Project 2

This project explores the use of Hermite interpolation in the context solving for the roots of $f(x) = 0$ and the difficulties which happen with a repeated root.

For this project the class has not been randomly grouped into teams, instead, please form your own teams consisting of 3 or 4 students. Each team should present their work in the form of a typed report using clear and properly punctuated English. Pencil and paper calculations may be typed or hand written. Where appropriate include full program listings and output.

Every team member should participate in the work and be prepared to independently answer questions concerning the material. One report per team must be submitted through WebCampus to complete this project. Make sure the report addresses each of the items listed below and then upload the report as a single pdf file.

1. List the members on your team and explain how the work for this project was conducted. Please provide details concerning how many meetings were held, what was discussed at each meeting and what work was done between meetings.

The answers will vary for this question. Please see my example solution the Programming Project 1 for what the answer is expected to look like.

Further include a written statement attesting that the submitted report represents the original efforts of the team members listed and, in particular, does contain the work of other students in the class.

The report you are reading here represents the independent work of the team members listed above and, in particular, does not contain the work of any other student in the class.

(i) It is fine to consult books, published papers and online resources while working on this project. If you do so, every outside source of information should be cited with a proper bibliographic reference at the point it is used.

(ii) There is no need to cite any additional help provided by the instructor during class or individually during office hours.

2. Let x_1 and x_2 be distinct. The corresponding Lagrange basis functions are given by

$$\ell_1(x) = \frac{x - x_2}{x_1 - x_2} \qquad \text{and} \qquad \ell_2(x) = \frac{x - x_1}{x_2 - x_1}.$$

Consider the following polynomials $\eta_1(x) = \ell_1(x)^2\ell_2(x)$ and $\eta_2(x) = \ell_1(x)\ell_2(x)^2$. Use calculus to find $\eta_i(x_j)$ and $\eta_i'(x_j)$ for $i = 1, 2$ and $j = 1, 2$.

First note that

$$\eta_1(x_1) = \ell_1(x_1)^2\ell_2(x_1) = 1^2 \cdot 0 = 0 \quad \textit{and} \quad \eta_1(x_2) = \ell_1(x_2)^2\ell_2(x_2) = 0^2 \cdot 1 = 0,$$

while

$$\eta_2(x_1) = \ell_1(x_1)\ell_2(x_1)^2 = 1 \cdot 0^2 = 0 \quad \textit{and} \quad \eta_2(x_2) = \ell_1(x_2)\ell_2(x_2)^2 = 0 \cdot 1^2 = 0.$$

Now, using the product and power rules it follows that

$$\eta_1'(x) = \frac{d}{dx}\big(\ell_1(x)^2\ell_2(x)\big) = 2\ell_1(x)\ell_1'(x)\ell_2(x) + \ell_1(x)^2\ell_2'(x)$$

and

$$\eta_2'(x) = \frac{d}{dx}\big(\ell_1(x)\ell_2(x)^2\big) = \ell_1'(x)\ell_2(x)^2 + 2\ell_1(x)\ell_2(x)\ell_2'(x).$$

Therefore

$$\begin{aligned}\eta_1'(x_1) &= 2\ell_1(x_1)\ell_1'(x_1)\ell_2(x_1) + \ell_1(x_1)^2\ell_2'(x_1)\\ &= 2 \cdot 1 \cdot \ell_1'(x_1) \cdot 0 + 1^2 \cdot \ell_2'(x_1) = 1/(x_2 - x_1),\end{aligned}$$

$$\begin{aligned}\eta_1'(x_2) &= 2\ell_1(x_2)\ell_1'(x_2)\ell_2(x_2) + \ell_1(x_2)^2\ell_2'(x_2)\\ &= 2 \cdot 0 \cdot \ell_1'(x_2) \cdot 1 + 0^2 \cdot \ell_2'(x_2) = 0,\end{aligned}$$

$$\begin{aligned}\eta_2'(x_1) &= \ell_1'(x_1)\ell_2(x_1)^2 + 2\ell_1(x_1)\ell_2(x_1)\ell_2'(x_1)\\ &= \ell_1'(x_1) \cdot 0^2 + 2 \cdot 1 \cdot 0 \cdot \ell_2'(x_1) = 0\end{aligned}$$

and

$$\begin{aligned}\eta_2'(x_2) &= \ell_1'(x_2)\ell_2(x_2)^2 + 2\ell_1(x_2)\ell_2(x_2)\ell_2'(x_2)\\ &= \ell_1'(x_2) \cdot 1^2 + 2 \cdot 0 \cdot 1 \cdot \ell_2'(x_2) = 1/(x_1 - x_2).\end{aligned}$$

3. Let A and B be constants and define $g_1(x) = A\eta_1(x)$ and $g_2(x) = B\eta_2(x)$. Solve for A and B such that

$$g_1(x_1) = 0, \quad g_1(x_2) = 0, \quad g_1'(x_1) = 1 \quad \text{and} \quad g_1'(x_2) = 0$$

and

$$g_2(x_1) = 0, \quad g_2(x_2) = 0, \quad g_2'(x_1) = 0 \quad \text{and} \quad g_2'(x_2) = 1.$$

Since η_1 and η_2 and their derivatives are already zero for the values indicated, solving for A and B is possible. In particular,

$$g_1'(x_1) = A\eta_1'(x_1) = A\ell_2'(x_1) = 1 \quad \textit{implies} \quad A = 1/\ell_2'(x_1)$$

and

$$g_2'(x_2) = B\eta_2'(x_2) = B\ell_1'(x_2) = 1 \quad \textit{implies} \quad B = 1/\ell_1'(x_2).$$

Since

$$\ell_1'(x) = \frac{d}{dx}\frac{x - x_2}{x_1 - x_2} = \frac{1}{x_1 - x_2}$$

and

$$\ell_2'(x) = \frac{d}{dx}\frac{x - x_1}{x_2 - x_1} = \frac{1}{x_2 - x_1},$$

then $\ell_1'(x_2) = -\ell_2'(x_1)$ *and we obtain that*

$$A = x_2 - x_1 \qquad \textit{and} \qquad B = x_1 - x_2.$$

4. Let a and b be constants and consider the polynomials

$$h_1(x) = \ell_1(x) + a\big(g_1(x) + g_2(x)\big) \quad \text{and} \quad h_2(x) = \ell_2(x) + b\big(g_1(x) + g_2(x)\big).$$

Solve for a and b such that

$$h_1(x_1) = 1, \quad h_1(x_2) = 0, \quad h_1'(x_1) = 0 \quad \text{and} \quad h_1'(x_2) = 0$$

and

$$h_2(x_1) = 0, \quad h_2(x_2) = 1, \quad h_2'(x_1) = 0 \quad \text{and} \quad h_2'(x_2) = 0.$$

The conditions for h_1 *given by*

$$h_1(x_1) = \ell_1(x_1) + a\big(g_1(x_1) + g_2(x_1)\big) = 1 + a(0 - 0) = 1$$

and

$$h_1(x_2) = \ell_1(x_2) + a\big(g_1(x_2) + g_2(x_2)\big) = 0 + a(0 - 0) = 0$$

are automatically satisfied for any value of a*. Now consider the derivative conditions*

$$h_1'(x_1) = \ell_1'(x_1) + a\big(g_1'(x_1) + g_2'(x_1)\big) = \ell_1'(x_1) + a(1 + 0) = \frac{1}{x_1 - x_2} + a = 0$$

and

$$h_1'(x_2) = \ell_1'(x_2) + a\big(g_1'(x_2) + g_2'(x_2)\big) = \ell_1'(x_2) + a(0 + 1) = \frac{1}{x_1 - x_2} + a = 0.$$

Fortunately, these are the same equation written twice, so it is possible to solve for a *as*

$$a = \frac{1}{x_2 - x_1}$$

to obtain that both $h_1'(x_1) = 0$ *and* $h_1'(x_2) = 0$*.*

We now perform similar algebra to solve for b*. As before the conditions on* h_2 *are automatically satisfied for any value of* b *which leaves the derivative conditions*

$$h_2'(x_1) = \ell_2'(x_1) + b\big(g_1'(x_1) + g_2'(x_1)\big) = \ell_2'(x_1) + b(1 + 0) = \frac{1}{x_2 - x_1} + b = 0$$

and

$$h_2'(x_2) = \ell_2'(x_2) + b\big(g_1'(x_2) + g_2'(x_2)\big) = \ell_2'(x_2) + b(0+1) = \frac{1}{x_2 - x_1} + b = 0$$

In summary we have

$$a = \frac{1}{x_2 - x_1} \qquad \textit{and} \qquad b = \frac{1}{x_1 - x_2}.$$

5. Let y_i and z_i for $i = 1, 2$ be constants. Show that the polynomial given by

$$p(x) = \sum_{i=1}^{2} \Big\{ y_i h_i(x) + z_i g_i(x) \Big\}$$

satisfies the conditions $p(x_i) = y_i$ and $p'(x_i) = z_i$ for $i = 1, 2$ and explain why $p(x)$ is the unique polynomial of minimal degree that satisfies those conditions.

By definition

$$\begin{aligned} p(x_1) &= y_1 h_1(x_1) + z_1 g_1(x_1) + y_2 h_2(x_1) + z_2 g_2(x_1) \\ &= y_1 \cdot 1 + z_1 \cdot 0 + y_2 \cdot 0 + z_2 \cdot 0 = y_1, \end{aligned}$$

$$\begin{aligned} p(x_2) &= y_1 h_1(x_2) + z_1 g_1(x_2) + y_2 h_2(x_2) + z_2 g_2(x_2) \\ &= y_1 \cdot 0 + z_1 \cdot 0 + y_2 \cdot 1 + z_2 \cdot 0 = y_2, \end{aligned}$$

$$\begin{aligned} p'(x_1) &= y_1 h_1'(x_1) + z_1 g_1'(x_1) + y_2 h_2'(x_1) + z_2 g_2'(x_1) \\ &= y_1 \cdot 0 + z_1 \cdot 1 + y_2 \cdot 0 + z_2 \cdot 0 = z_1 \end{aligned}$$

and

$$\begin{aligned} p'(x_2) &= y_1 h_1'(x_2) + z_1 g_1'(x_2) + y_2 h_2'(x_2) + z_2 g_2'(x_2) \\ &= y_1 \cdot 0 + z_1 \cdot 0 + y_2 \cdot 0 + z_2 \cdot 1 = z_2. \end{aligned}$$

Thus, $p(x_i) = y_i$ and $p'(x_i) = z_i$ for $i = 1, 2$.

Since $\ell_i(x)$ is a polynomial of degree 1 for $i = 1, 2$, it follows that $\eta_1(x) = \ell_1(x)^2 \ell_2(x)$ and $\eta_2 = \ell_1(x)\ell_2^2(x)$ are polynomials of degree three. Consequently, $g_i(x)$, $h_i(x)$ and finally $p(x)$ are all polynomials of at most degree three.

Note that $p(x)$ may also be written as

$$p(x) = p_3 x^3 + p_2 x^2 + p_1 x + p_0$$

where the p_i represent four independent parameters. Now, since the conditions $p(x_i) = y_i$ and $p'(x_i) = z_i$ for $i = 1, 2$ represent four independent linear equations in the p_i, there is exactly one choice which satisfies these conditions. In particular, the polynomial p must be unique, because it is polynomial of degree three and the number of parameters needed to specify a polynomial of degree three exactly matches the number of independent conditions used to determine that polynomial.

6. Let $f(x) = e^x - 13/x$ and define $y_i = f(x_i)$ and $z_i = f'(x_i)$ for $i = 1, 2$ where $x_1 = 1.3$ and $x_2 = 2.7$. Write a computer program that computes $p(x)$ for any value of x, verify $p(2) \approx 0.9393631693832503$ and then compute the size of the error $e = |f(2) - p(2)|$.

The Julia program

```
x1=1.3
x2=2.7
l1(x)=(x-x2)/(x1-x2)
l2(x)=(x-x1)/(x2-x1)
eta1(x)=l1(x)^2*l2(x)
eta2(x)=l1(x)*l2(x)^2
g1(x)=(x2-x1)*eta1(x)
g2(x)=(x1-x2)*eta2(x)
h1(x)=l1(x)+(g1(x)+g2(x))/(x2-x1)
h2(x)=l2(x)+(g1(x)+g2(x))/(x1-x2)
f(t)=exp(t)-13/t
df(t)=exp(t)+13/t^2
y=f.([x1,x2])
z=df.([x1,x2])
function p(x)
    h=[h1,h2]
    g=[g1,g2]
    r=0
    for i=1:2
        r=r+y[i]*h[i](x)+z[i]*g[i](x)
    end
    return r
end
println("The value of p(2) = ",p(2))
println("  e = |f(2)-p(2)| = ",abs(f(2)-p(2)))
```

produces the output

```
The value of p(2) = 0.9393631693832503
  e = |f(2)-p(2)| = 0.050307070452599856
```

7. Plot $f(x)$ and $p(x)$ on the same graph over the interval $[0.5, 4]$. You might find the plotting commands

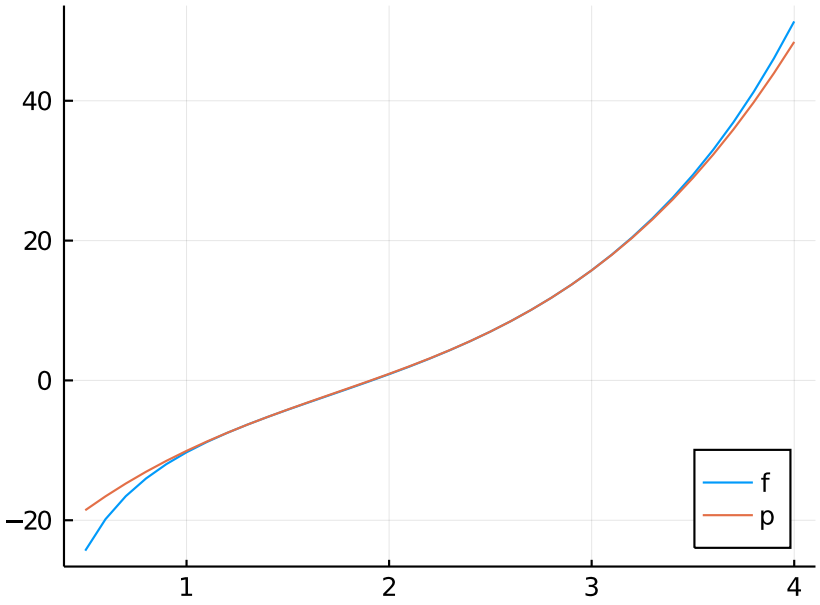
```
xs=[0.5:0.1:4;];
yf=f.(xs);
yp=p.(xs);
plot(xs,[yf yp],size=[400,300],
    legend=:bottomright,label=["f" "p"])
```

to be useful when making the graph in Julia.

I added the lines

```
using Plots
xs=[0.5:0.1:4;];
yf=f.(xs);
yp=p.(xs);
plot(xs,[yf yp],size=[400,300],
    legend=:bottomright,label=["f" "p"])
savefig("herm7.pdf")
```

to the code in the previous question and obtained the graph



8. [Extra Credit] Modify the program in Question 6 to estimate the maximum error

$$E = \max\{\, |f(x) - p(x)| : x \in [1,3] \,\}$$

good to 4 significant digits. Use any reasonable method you can find to compute E and explain how it works.

To compute the maximum value I divided the interval $[1,3]$ *up into* 1001 *points of the form* $x_i = 1 + i * h$ *where* $h = 2/1000$ *and* $i = 0, \ldots 1000$ *and then approximated* E *as*

$$E \approx \max\{\, |f(x_i) - p(x_i)| : i = 0, \ldots, 1000 \,\}.$$

The modifications needed to compute this involved replacing the last two lines of the code from Problem 6 with

```
xs=[1:2/1000:3;];
println("  E = ",maximum(abs.(f.(xs)-p.(xs))))
```

to obtain the output

```
E = 0.20574469650148863
```

A more accurate approximation could be obtained by choosing a smaller size for h or using calculus to check only the endpoints of the interval along with the critical points in that interval where the derivative $f'(x) - p'(x) = 0$.

9. Let $f\colon \mathbf{R} \to \mathbf{R}$ be continuously differentiable such that $f'(x) \neq 0$ for $x \in \mathbf{R}$. By the inverse function theorem f is invertible and $(f^{-1})'(y) = 1/f'(f^{-1}(y))$. We may use this formula to adapt the polynomial developed above to interpolate the inverse function f^{-1} by swapping the roles of x_i, y_i and z_i as

original	replacement
x_i	y_i
y_i	x_i
z_i	$1/z_i$

Thus, everywhere there used to be x_i write y_i, everywhere there used to be y_i write x_i and replace z_i by $1/z_i$. Let $q(y)$ be the polynomial obtained in this way which interpolates $f^{-1}(y)$ with $x_1 = 1.3$ and $x_2 = 2.7$ where $f(x)$ is the function from Question 6. Verify that $q(0) \approx 1.8849155134426776$.

Swapping the variables as suggested led to the following code for the inverse function

```
f(t)=exp(t)-13/t
df(t)=exp(t)+13/t^2
x=[1.3,2.7]
y1=f(x[1])
y2=f(x[2])
z=df.(x)
l1(y)=(y-y2)/(y1-y2)
l2(y)=(y-y1)/(y2-y1)
eta1(y)=l1(y)^2*l2(y)
eta2(y)=l1(y)*l2(y)^2
g1(y)=(y2-y1)*eta1(y)
g2(y)=(y1-y2)*eta2(y)
h1(y)=l1(y)+(g1(y)+g2(y))/(y2-y1)
h2(y)=l2(y)+(g1(y)+g2(y))/(y1-y2)
function q(y)
    h=[h1,h2]
    g=[g1,g2]
    r=0
    for i=1:2
        r=r+x[i]*h[i](y)+(1/z[i])*g[i](y)
    end
    return r
end
println("The value of q(0) = ",q(0))
```

and the output

```
The value of q(0) = 1.8849155134426772
```

Note the slight rounding difference in the last digit is not significant.

10. Consider the two-step Newton-like method for approximating the root of $f(x) = 0$ obtained by iterating the inverse interpolation in Question 9 as follows:

Inverse Hermite Method. Let $q_n(y)$ be the polynomial such that

$$q_n(y_i) = x_i \quad \text{and} \quad q_n'(y_i) = 1/z_i \qquad \text{for} \qquad i = n, n+1.$$

Define

$$x_{n+2} = q_n(0), \quad y_{n+2} = f(x_{n+2}) \quad \text{and} \quad z_{n+2} = f'(x_{n+2}).$$

Now repeat until x_n converges.

Note that $x_3 \approx 1.8849155134426776$ has already been computed above. Write a program to perform five iterations of the inverse Hermite method for the function $f(x)$ given in Question 6 starting with $x_1 = 1.3$ and $x_2 = 2.7$. Report the values of $x_3, \ldots, x_7$. Did the method converge? If so, to what did it converge?

I replaced the last line of the code from Problem 9 with the loop

```
for i=3:7
    global y1,y2,x,z
    x3=q(0)
    println("x",i," = ",x3)
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=f(x3)
    z[1]=z[2]; z[2]=df(x3)
end
```

and obtained the output

```
x3 = 1.8849155134426772
x4 = 1.915096234662282
x5 = 1.9151522395359102
x6 = 1.9151522395363563
x7 = 1.9151522395363563
```

It appears the method converged to 1.9151522395363563.

11. [Extra Credit] Theoretically, what should the order of convergence be for the inverse Hermite method? Use big-number arithmetic to verify your prediction.

The inverse Hermite interpolation is discussed in details in Section 8.5-2 of the text, Ralston and Rabinowitz, A First Course in Numerical Analysis. *Let α be the root such that*

$f(\alpha) = 0$. From a generalization of the theorem on interpolating polynomials we obtain that the error $e_n = x_n - \alpha$ satisfies

$$|e_{n+1}| \approx K|e_n|^2|e_{n-1}|^2$$

Heuristically substituting $|e_n| = cd^{\gamma_n}$ into the above yields

$$|cd^{\gamma_{n+1}}| \approx K|cd^{\gamma_n}|^2|cd^{\gamma_{n-1}}|^2$$

from which it may be inferred that

$$c = Kc^4 \qquad \text{and} \qquad \gamma_{n+1} = 2(\gamma_n + \gamma_{n-1}).$$

Consequently $c = K^{1/3}$ and we solve the difference equation for γ_n by substituting $\gamma_n = \rho^n$ to obtain

$$\rho^{n+1} = 2\rho^n + 2\rho^{n-1} \qquad \text{or that} \qquad \rho^2 - 2\rho - 2 = 0.$$

Solving this quadratic equation for ρ yields that

$$\rho = \frac{2 \pm \sqrt{4 - 4 \cdot 1 \cdot (-2)}}{2} = 1 \pm \sqrt{3}.$$

In particular

$$\gamma_n = c_1(1 + \sqrt{3})^n + c_2(1 - \sqrt{3})^n$$

where, as with the secant method, it is convenient to set $c_1 = 1/2$ and $c_2 = 1/2$ such that $\gamma_0 = 1$ and $\gamma_1 = 1$. At any rate, the rate of convergence is dominated by the the larger term, and so we expect that

$$\lim_{n\to\infty} \frac{|e_{n+1}|}{|e_n|^{1+\sqrt{3}}} = \frac{cd^{\gamma_{n+1}}}{(cd^{\gamma_n})^{1+\sqrt{3}}} = \frac{K^{1/3}}{K^{(1+\sqrt{3})/3}} \frac{d^{c_1(1+\sqrt{3})^{n+1}}}{(d^{c_1(1+\sqrt{3})^n})^{1+\sqrt{3}}} = \frac{1}{K^{1/\sqrt{3}}}$$

which suggests the theoretical order of convergence should be

$$1 + \sqrt{3} \approx 2.732050807568877.$$

For numerical confirmation, we follow the same approach that was used to verify the order of convergence for the secant method. The only difference is that in this case we don't know the exact value to which the method should be converging. Therefore, we first approximate $\alpha = x_{12}$ using 4096-bit extended precision arithmetic and then approximate the order of the convergence as

$$\text{order} = \lim_{n\to\infty} \frac{\log|e_n| - \log|e_{n-1}|}{\log|e_{n-1}| - \log|e_{n-2}|} \approx \frac{\log|e_9| - \log|e_8|}{\log|e_8| - \log|e_7|} \approx 2.7331765471982905.$$

This agrees reasonably well with the theoretical prediction of $1 + \sqrt{3}$.

For reference the Julia code was

```
setprecision(4096)
f(t)=exp(t)-13/t
df(t)=exp(t)+13/t^2
x=[big(1.3),big(2.7)]
y1=f(x[1])
y2=f(x[2])
z=df.(x)
l1(y)=(y-y2)/(y1-y2)
l2(y)=(y-y1)/(y2-y1)
eta1(y)=l1(y)^2*l2(y)
eta2(y)=l1(y)*l2(y)^2
g1(y)=(y2-y1)*eta1(y)
g2(y)=(y1-y2)*eta2(y)
h1(y)=l1(y)+(g1(y)+g2(y))/(y2-y1)
h2(y)=l2(y)+(g1(y)+g2(y))/(y1-y2)
function q(y)
    h=[h1,h2]
    g=[g1,g2]
    r=0
    for i=1:2
        r=r+x[i]*h[i](y)+(1/z[i])*g[i](y)
    end
    return r
end
n=12
xs=big.(zeros(n))
xs[1]=x[1]; xs[2]=x[2]
for i=3:n
    global y1,y2,x,z
    x3=q(0)
    xs[i]=x3
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=f(x3)
    z[1]=z[2]; z[2]=df(x3)
end
alpha=xs[n];
n=9
e=xs[1:n].-alpha;
lek=log.(abs.(e));
slope=(lek[n]-lek[n-1])/(lek[n-1]-lek[n-2]);
println("Numerically the order is ",Float64(slope))
```

and the corresponding output

```
Numerically the order is 2.7331765471982905
```

12. By definition, if α is a root of higher multiplicity, then $f(\alpha) = 0$ and also $f'(\alpha) = 0$. Let $f(x) = x^4 - 4x^2 + 4$ and use calculus to show this function has a root $\alpha \in [1, 2]$ of higher multiplicity. What is the exact value of α?

First note that

$$f'(x) = 4x^3 - 8x = 4x(x^2 - 2) = 4x(x - \sqrt{2})(x + \sqrt{2}).$$

Solving $f'(x) = 0$ *yields the possible choices for* α *of*

$$\alpha = -\sqrt{2}, \qquad \alpha = 0 \qquad \textit{and} \qquad \alpha = \sqrt{2}.$$

As only $\sqrt{2}$ *is in the interval* $[1, 2]$ *we check to make sure this is also a root of* $f(x)$*. The easy calculation*

$$f(\sqrt{2}) = (\sqrt{2})^4 - 4(\sqrt{2})^2 + 4 = 4 - 8 + 4 = 0$$

shows this is indeed true. Thus $\alpha = \sqrt{2}$ *is a root in the interval* $[1, 2]$ *of higher multiplicity.*

13. When $f(x)$ has a root α of higher multiplicity, then it may not be invertible near $x = \alpha$ and consequently approximating the inverse function may not make sense. Moreover, since $z_n \to 0$ as $x_n \to \alpha$ there could be numerical difficulties when calculating the quantities $1/z_n$. Let $f(x)$ be as in Question 12 and perform twenty steps of the inverse Hermite method starting with $x_1 = 1$ and $x_2 = 2$ to find $x_3, \ldots, x_{22}$.

Simple modifications to the program from Problem 10 led to the code

```
f(t)=(t^2-4)*t^2+4
df(t)=4*t*(t^2-2)
x=[1.0,2.0]
y1=f(x[1])
y2=f(x[2])
z=df.(x)
l1(y)=(y-y2)/(y1-y2)
l2(y)=(y-y1)/(y2-y1)
eta1(y)=l1(y)^2*l2(y)
eta2(y)=l1(y)*l2(y)^2
g1(y)=(y2-y1)*eta1(y)
g2(y)=(y1-y2)*eta2(y)
h1(y)=l1(y)+(g1(y)+g2(y))/(y2-y1)
h2(y)=l2(y)+(g1(y)+g2(y))/(y1-y2)
function q(y)
    h=[h1,h2]
    g=[g1,g2]
    r=0
    for i=1:2
        r=r+x[i]*h[i](y)+(1/z[i])*g[i](y)
    end
```

```
    return r
end
for i=3:22
    global y1,y2,x,z
    x3=q(0)
    println("x",i," = ",x3)
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=f(x3)
    z[1]=z[2]; z[2]=df(x3)
end
```

with the output

```
x3 = 1.824074074074074
x4 = 1.5867229698031553
x5 = 1.4930710039744846
x6 = 1.4482555951243439
x7 = 1.4288298784998776
x8 = 1.4204425367437337
x9 = 1.416862544332086
x10 = 1.4153387774900275
x11 = 1.4146913139335793
x12 = 1.414416368936844
x13 = 1.4142996471626235
x14 = 1.4142501012728852
x15 = 1.4142290711684806
x16 = 1.4142201449795146
x17 = 1.4142163563111525
x18 = 1.41421474824092
x19 = 1.414214065715211
x20 = 1.4142137760132185
x21 = 1.4142136530857639
x22 = 1.4142136009666249
```

14. Compute the errors $e_n = |x_n - \alpha|$ for $n = 3, \ldots 22$ where α is the root found earlier. Does the method converge? If so, what is the order of convergence?

After changing the last lines in the program from Problem 13 as

```
alpha=sqrt(2.0)
n=22
xs=zeros(n);
xs[1]=x[1]
xs[2]=x[2]
for i=3:n
    global y1,y2,x,z
    x3=q(0)
```

```
    xs[i]=x3
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=f(x3)
    z[1]=z[2]; z[2]=df(x3)
end
using Printf
e=xs.-alpha;
lek=log.(abs.(e));
@printf("%6s %23s %19s\n","n","e[n]","slope[n]");
for i=3:n
    slope=(lek[i]-lek[i-1])/(lek[i-1]-lek[i-2]);
    @printf("%6d %23.15e %19.15f\n",i,e[i],slope);
end
```

I obtained the output

```
     n                    e[n]            slope[n]
     3   4.098605117009788e-01  -1.030483585046249
     4   1.725094074300602e-01   2.423052592017294
     5   7.885744160138941e-02   0.904600926276018
     6   3.404203275124873e-02   1.073115539874145
     7   1.461631612678249e-02   1.006442392185226
     8   6.228974370638518e-03   1.008834488405761
     9   2.648981958990770e-03   1.002473290983210
    10   1.125215116932310e-03   1.001362369503412
    11   4.777515604841298e-04   1.000510971780564
    12   2.028065637489362e-04   1.000232946104958
    13   8.608478952831788e-05   1.000095337609654
    14   3.653889979005065e-05   1.000041302599187
    15   1.550879538547711e-05   1.000017315955380
    16   6.582606419502923e-06   1.000007461614518
    17   2.793938057354950e-06   1.000002521260354
    18   1.185867824826659e-06   0.999998312611732
    19   5.033421157651929e-07   0.999979923746770
    20   2.136401233698848e-07   1.000019976376144
    21   9.071266870996908e-08   0.999555124562386
    22   3.859352970536634e-08   0.997684413085831
```

The second column labeled `e[n]` *shows that the method still converges while the third column labeled* `slope[n]` *represents the quotient*

$$\texttt{slope[n]} = \frac{\log|e_n| - \log|e_{n-1}|}{\log|e_{n-1}| - \log|e_{n-2}|}$$

and indicates a linear order of convergence.

15. [Extra Credit] When $f(x)$ has a root of higher multiplicity, the simple trick of setting $r(x) = f(x)/f'(x)$ and then searching for the roots of $r(x)$ often leads to fast convergence. Use the inverse Hermite method to solve $r(x) = 0$ where $f(x)$ is the same function as in Question 12. Did you get the same root? Was the convergence faster? Can you think of anything that might make this idea work better?

Differentiating by the quotient rule yields

$$r'(x) = \frac{d}{dx}\frac{f(x)}{f'(x)} = \frac{f'(x)^2 - f(x)f''(x)}{f'(x)^2} = 1 - \frac{f(x)f''(x)}{f'(x)^2}.$$

Replacing f by r in the previous program led to

```
f(t)=(t^2-4)*t^2+4
df(t)=4*t*(t^2-2)
ddf(t)=12*t^2-8
r(t)=f(t)/df(t)
dr(t)=1-f(t)*ddf(t)/df(t)^2
x=[1.0,2.0]
y1=r(x[1])
y2=r(x[2])
z=df.(x)
l1(y)=(y-y2)/(y1-y2)
l2(y)=(y-y1)/(y2-y1)
eta1(y)=l1(y)^2*l2(y)
eta2(y)=l1(y)*l2(y)^2
g1(y)=(y2-y1)*eta1(y)
g2(y)=(y1-y2)*eta2(y)
h1(y)=l1(y)+(g1(y)+g2(y))/(y2-y1)
h2(y)=l2(y)+(g1(y)+g2(y))/(y1-y2)
function q(y)
    h=[h1,h2]
    g=[g1,g2]
    s=0
    for i=1:2
        s=s+x[i]*h[i](y)+(1/z[i])*g[i](y)
    end
    return s
end
alpha=sqrt(2.0)
n=10
xs=zeros(n);
xs[1]=x[1]
xs[2]=x[2]
for i=3:n
    global y1,y2,x,z
```

```
    x3=q(0)
    xs[i]=x3
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=r(x3)
    z[1]=z[2]; z[2]=dr(x3)
end
using Printf
e=xs.-alpha;
lek=log.(abs.(e));
@printf("%6s %19s %23s %19s\n","n","x[n]","e[n]","slope[n]");
for i=3:n
    slope=(lek[i]-lek[i-1])/(lek[i-1]-lek[i-2]);
    @printf("%6d %19.15f %23.15e %19.15f\n",i,xs[i],e[i],slope);
end
```

with output

```
     n                x[n]                     e[n]            slope[n]
     3   1.480468750000000   6.625518762690485e-02  -6.288538935926606
     4   1.428670635400296   1.445707302720134e-02   0.698495452311242
     5   1.414213600573225   3.820012994815158e-08   8.436971957382879
     6   1.414213563199658   8.265630402348734e-10   0.298455400148358
     7   1.414213563199658   8.265630402348734e-10  -0.000000000000000
     8                 NaN                     NaN                 NaN
     9                 NaN                     NaN                 NaN
    10                 NaN                     NaN                 NaN
```

The same root was obtained to machine precision in only 6 iterations. This is much faster convergence. The only thing I could think of (it being late at night) to improve things was to use extended precision arithmetic to numerically verify the order of convergence. These modifications led to the code

```
setprecision(8192)
f(t)=(t^2-4)*t^2+4
df(t)=4*t*(t^2-2)
ddf(t)=12*t^2-8
r(t)=f(t)/df(t)
dr(t)=1-f(t)*ddf(t)/df(t)^2
x=[big(1.0),big(2.0)]
y1=r(x[1])
y2=r(x[2])
z=df.(x)
l1(y)=(y-y2)/(y1-y2)
l2(y)=(y-y1)/(y2-y1)
eta1(y)=l1(y)^2*l2(y)
eta2(y)=l1(y)*l2(y)^2
```

```
g1(y)=(y2-y1)*eta1(y)
g2(y)=(y1-y2)*eta2(y)
h1(y)=l1(y)+(g1(y)+g2(y))/(y2-y1)
h2(y)=l2(y)+(g1(y)+g2(y))/(y1-y2)
function q(y)
    h=[h1,h2]
    g=[g1,g2]
    s=0
    for i=1:2
        s=s+x[i]*h[i](y)+(1/z[i])*g[i](y)
    end
    return s
end
alpha=sqrt(big(2))
n=10
xs=big.(zeros(n));
xs[1]=x[1]
xs[2]=x[2]
for i=3:n
    global y1,y2,x,z
    x3=q(0)
    xs[i]=x3
    x[1]=x[2]; x[2]=x3
    y1=y2; y2=r(x3)
    z[1]=z[2]; z[2]=dr(x3)
end
using Printf
e=xs.-alpha;
lek=log.(abs.(e));
@printf("%6s %23s %19s\n","n","e[n]","slope[n]");
for i=3:n
    slope=(lek[i]-lek[i-1])/(lek[i-1]-lek[i-2]);
    @printf("%6d %23.15e %19.15f\n",i,e[i],slope);
end
```

with output

```
     n                    e[n]            slope[n]
     3   6.625518762690495e-02  -6.288538935926595
     4   1.445707302720101e-02   0.698495452311254
     5   3.820013453116275e-08   8.436971878573467
     6   1.334081199462426e-20   2.233210479208672
     7   1.147781764040703e-56   2.895212551902124
     8  1.036210975814177e-153   2.690795567926194
     9  6.251439558121787e-420   2.743274600211047
```

```
10 1.854477074832233e-1146   2.729055705851005
```

The result was over 1000 digits of precision in 10 iterations with an effective order of convergence around 2.729055705851005. Thus, the theoretical order of $1 + \sqrt{3}$ *for the inverse Hermite interpolation method appears to have been restored when finding roots of higher multiplicity by using* $r(x)$ *in place of* $f(x)$*.*