The computer labs provide computational experience related to the analytic theory presented in the lectures. The main tool for these exercises is a programming language called Julia designed for the implementation of numerical algorithms that combines the compiled efficiency of C and Fortran with the interactive and notational convenience of MATLAB and Python.

In science and engineering an important goal is to become a skilled practitioner by doing it yourself. To this end the computers in the lab have been provisioned with a Linux programming environment similar to what is deployed on the university high-performance cluster, all other supercomputers worldwide and for most cloud computing. To access Linux please restart the computer using the USB network boot key for this class.

Rather than using the lab equipment it is also possible to freely install Julia on your personal laptop. While using your own computer goes along well with doing it yourself, I will unfortunately be unable to help with any technical problems that might crop up in that case. Even so, I'd recommend trying to install Julia at home, if only to avoid coming in after hours to complete the homework. You may also use your laptop in the lab.

Note that it is possible to forgo Julia and perform all your computations using a different programming language. Although I would be happy to grade assignments completed using such alternatives, my opinion is Julia makes numerical methods much easier than a general-purpose programming language. I am also able to provide more help with Julia.

Newton's Method

The first computer lab introduced Julia by using it as a desktop calculator. The next activity is to approximate solutions to $f(x) = 0$ with Newton's method and verify the quadratic rate of convergence numerically. Note that

- Newton's method is one of the most important algorithms used in computation due to its fast rate of convergence.
- As it is covered in an introductory calculus course, the familiarity makes a foundation on which to build analysis.

Newton's method described algorithmically is the iterative scheme

$$x_{n+1} = g(x_n) \qquad \text{where} \qquad g(x) = x - \frac{f(x)}{f'(x)}.$$

Here x_0 is an initial guess for the solution and each x_n is an improved approximation of that solution.

The goal today is for you to write a program that approximates the solution near $x_0 = 1$ to the equation

$$\cos x = 2x \tag{3.1}$$

using Newton's method. Setting $f(x) = \cos x - 2x$ converts the task of solving this equation into finding the root ξ such that $f(\xi) = 0$.

How to find the roots of f will now be described in step-by-step details. We focus on a similar example so as to not so spoil the fun of doing the problem yourself.

Suppose then instead of (3.1) that one seeks to approximate the solution near $x_0 = 2$ to the equation

$$5xe^{-x} = 1 \tag{3.2}$$

using Newton's method. Note that the roots of the function

$$f(x) = 5xe^{-x} - 1$$

correspond to the solutions of equation (3.2).

One of the complications of Newton's method is that the algorithm requires the derivative $f'(x)$ to perform the iterations. While it might appear necessary to find the derivative by hand, in the 1960s computer algebra systems began to appear that were capable of finding such derivatives using the rules of calculus.

Such computer algebra systems are now widely available. One is even built into Julia. For this lab, however, we simply find f' by hand as

$$f'(x) = 5e^{-x} - 5xe^{-x} = 5(1-x)e^{-x}.$$

Open the Julia REPL and enter

```
f(x)=5*x*exp(-x)-1
df(x)=5*(1-x)*exp(-x)
x0=2
g(x)=x-f(x)/df(x)
```

At this point the screen should look like

```
julia> f(x)=5*x*exp(-x)-1
f (generic function with 1 method)

julia> df(x)=5*(1-x)*exp(-x)
df (generic function with 1 method)

julia> g(x)=x-f(x)/df(x)
g (generic function with 1 method)

julia> x0=2
2
```

To test Newton's method set `xn=x0` and compute

```
xn=g(xn)
```

as many times as needed. In the REPL this looks like

```
julia> xn=x0
2

julia> xn=g(xn)
2.5221887802138703

julia> xn=g(xn)
2.5425691666820014

julia> xn=g(xn)
2.5426413568569757

julia> xn=g(xn)
2.5426413577735265

julia> xn=g(xn)
2.5426413577735265
```

Note that the up-arrow key followed by ⟨enter⟩ was repeatedly pressed to iterate the recurrence. As the value didn't change after the last iteration shown we conclude the method has converged to all available digits.

Checking the Order of Convergence

While it only took a few iterations to find a good approximation, was the order of convergence really quadratic?

Quadratic convergence means for some $\mu > 0$ that

$$\lim_{n\to\infty} \frac{|x_{n+1}-\xi|}{|x_n-\xi|^2} = \mu$$

The difficulty is we don't know what ξ is. Since

$$|x_{n+1}-\xi| - |x_{n+2}-\xi| \le |x_{n+1}-x_{n+2}| \le |x_{n+1}-\xi| + |x_{n+2}-\xi|$$

and

$$|x_n-\xi| - |x_{n+1}-\xi| \le |x_n-x_{n+1}| \le |x_n-\xi| + |x_{n+1}-\xi|,$$

it follows that

$$\frac{|x_{n+1}-\xi| - |x_{n+2}-\xi|}{(|x_n-\xi| + |x_{n+1}-\xi|)^2} \le \frac{|x_{n+1}-x_{n+2}|}{|x_n-x_{n+1}|^2} \le \frac{|x_{n+1}-\xi| + |x_{n+2}-\xi|}{(|x_n-\xi| - |x_{n+1}-\xi|)^2}.$$

Under the assumption $|x_{n+1}-\xi| \approx \mu|x_n-\xi|^2$ we have

$$\begin{aligned}\lim_{n\to\infty} \frac{|x_{n+1}-\xi| - |x_{n+2}-\xi|}{(|x_n-\xi| + |x_{n+1}-\xi|)^2} &= \lim_{n\to\infty} \frac{|x_{n+1}-\xi| - \mu|x_{n+1}-\xi|^2}{(|x_n-\xi| + \mu|x_n-\xi|^2)^2}\\ &= \lim_{n\to\infty} \frac{|x_{n+1}-\xi|}{|x_n-\xi|^2} \lim_{n\to\infty} \frac{1-\mu|x_{n+1}-\xi|}{(1+\mu|x_n-\xi|)^2} = \mu\cdot 1 = \mu\end{aligned}$$

and similarly

$$\lim_{n\to\infty} \frac{|x_{n+1}-\xi| + |x_{n+2}-\xi|}{(|x_n-\xi| - |x_{n+1}-\xi|)^2} = \mu.$$

Therefore, by the squeezing theorem

$$\lim_{n\to\infty} \frac{|x_{n+1}-x_{n+2}|}{|x_n-x_{n+1}|^2} = \mu$$

and for large enough n we have

$$|x_{n+1}-x_{n+2}| \approx \mu |x_n - x_{n+1}|^2.$$

Consequently

$$\log|x_{n+1}-x_{n+2}| \approx 2\log|x_n - x_{n+1}| + \log\mu.$$

and so

$$\frac{\log|x_{n+1}-x_{n+2}| - \log\mu}{\log|x_n - x_{n+1}|} \approx 2.$$

Since $|x_{n+1}-x_{n+2}| \to 0$ then $\log|x_{n+1}-x_{n+2}| \to -\infty$ which means $\log\mu$ is negligible when n is large. Thus, for n large enough it follows that

$$\frac{\log|x_{n+1}-x_{n+2}|}{\log|x_n - x_{n+1}|} \approx 2.$$

We now compute this ratio using arbitrary precision arithmetic as a consistency check for the quadratic convergence of Newton's method.

Arbitrary Precision with Julia

Julia has a built-in data type called `BigFloat` which can be used to compute floating point numbers with thousands of digits. The default precision for `BigFloat` is 256 bits. That corresponds to approximately

$$256\log(2)/\log(10) \approx 77$$

decimal digits of precision. We'll set the precision to 4096 bits in order to examine the convergence over more iterations of Newton's method.

The function call `setprecision(4096)` sets the precision of the builtin `BigFloat` data type in Julia to 4096 bits. After this, specifying the starting value for x_0 with `x0=big"2.0"` causes the rest of the calculation to be performed using `BigFloat` numbers.

The following script `newtbig.jl` uses 4096-bit arithmetic to compute

$$\frac{\log\varepsilon_{n+1}}{\log\varepsilon_n} \qquad \text{where} \qquad \varepsilon_n = |x_n - x_{n+1}|$$

for $n = 0, 1, 2, 3, \ldots, 8$.

```julia
setprecision(4096)
f(x)=5*x*exp(-x)-1
df(x)=5*(1-x)*exp(-x)
x0=big"2.0"

g(x)=x-f(x)/df(x)

xn=x0
xn1=g(xn)
en=abs(xn-xn1)
for n=1:9
    global xn,xn1,en
    xn=xn1
    xn1=g(xn)
    en0=en
    en=abs(xn-xn1)
    println("n=$(n-1) log(e(n+1))/log(e(n))=",
        Float64(log(en)/log(en0)))
end
```

The output should look like

```
julia> include("newtbig.jl")
n=0 log(e(n+1))/log(e(n))=5.992036085626963
n=1 log(e(n+1))/log(e(n))=2.4494631630193666
n=2 log(e(n+1))/log(e(n))=2.18225176911099
n=3 log(e(n+1))/log(e(n))=2.083513514614094
n=4 log(e(n+1))/log(e(n))=2.0400830203421143
n=5 log(e(n+1))/log(e(n))=2.019647739794135
n=6 log(e(n+1))/log(e(n))=2.0097283003402056
n=7 log(e(n+1))/log(e(n))=2.004840604741725
n=8 log(e(n+1))/log(e(n))=2.002414458650865
```

Observe that the ratio is close to 2 as expected.

Submitting Your Work

For this lab two things should be uploaded for grading:

- A program that computes $\log \varepsilon_{n+1}/\log \varepsilon_n$ when $n = 0, 1, 2, \ldots, 8$ for the sequence x_n obtained by Newton's method corresponding to the solution of $f(x) = \cos x - 2x$ starting with $x_0 = 1$.
- The output from running that program.

The main thing needed to finish this lab is change the definitions of lines 2, 3 and 4 in the previous script. You may test your program by running it as

```
$ julia newtbig.jl
```

If everything looks fine place the output in `newtbig.out` and finally convert everything to Postscript format. Check the Postscript file using the `evince` previewer. A transcript of the commands needed to prepare and preview the program and output for submission look like

```
$ julia newtbig.jl >newtbig.out
$ j2pdf -o submit02.pdf newtbig.jl newtbig.out
$ evince submit02.pdf &
```

Upload `submit02.pdf` for grading to the course management system. Please reboot the lab computer into Microsoft Windows before leaving.