MTH 654, Fall 2009, LAB1

The goal of this assignment is to become familiar with the programming environment, to explore the concept of rate of convergence, memory limitations, and of computational complexity.

1. Consider a sequence of iterates $\{x_n\}$ converging to some number x. That is, the sequence of errors in the n-th iteration $\varepsilon_n := |x - x_n|$ converges to 0.

We say that this iteration has q-convergence rate α if $\varepsilon_{n+1} \leq C\varepsilon_n^\alpha$. C is called a factor[1] **Note:** with $\alpha = 1$, the convergence is *linear*, $\alpha = 2$, the convergence is *quadratic*. See [Kelley, Chapter 4.1] for additional definitions and conditions. **Explain** the conditions.

A good estimate for α can be provided by calculating

$$\alpha \approx \frac{\log(\varepsilon_{n+1}) - \log(\varepsilon_n)}{\log(\varepsilon_n) - \log(\varepsilon_{n-1})}$$

for a few subsequent iterations. When x_n is close to x, the calculated ratio should be close to α. **Explain** why.

Now define

- $a_k = 1 + \frac{1}{2^k}$.
- $b_k = 3 + \frac{1}{k!}$.
- $c_1 = 2$; for $k > 1 : c_k = \frac{c_{k-1}^2+1}{2c_{k-1}}$
- $d_1 = 2$; for $k > 1 : d_k = d_{k-1} - \frac{d_{k-1}^2-1}{4}$

Explore: for each sequence, find its limit and convergence rate, if any. **Explain** using calculus.

Hint for a) MATH: Note $a_{k+1} - 1 = \frac{1}{2}(a_k - 1)$.

Hint for a) MATLAB: you can set up example one as follows
`n=10;a=zeros(1,n);for j=1:n a(j)=1+1/(2ĵ);end;`

Choose some good n here: remember about underflow. (For some problems the convergence will be neither linear nor quadratic, see the text by [Kelley] to identify what kind it is).

2. **Explore:** The following formula comes from calculus

$$\frac{\pi}{4} = \sum_{j=1}^{\infty} (-1)^{j+1} \frac{1}{2j-1} = 1 - \frac{1}{3} + \frac{1}{5} - \ldots$$

Of course, on the computer we can only sum a finite number of terms, say N. To compute π, the algorithm can be implemented in two ways a) by setting up an array of values $a(j) = (-1)^{j+1}\frac{1}{2j-1}$ and then summing those, or b) without the

[1](Note: in some other sources q is called the order and C the rate. Sorry, I do not explain it, I just report it.)

array, by summing the values directly. Also, you have to consider multiplication by 4 (where should it go in the code ?).

i) Implement both variants of the algorithm and test its rate of convergence by considering $N = 10, 100, 1000, \ldots$ (what is the largest N you can use for a) and b) ?

ii) Discuss the complexity of the algorithm: time the execution of the loop in which you add numbers (not the one in which you set up the array). To get reasonable timings, perhaps that loop should be repeated. for example M=100 times. Let T_N be the computational time for one loop without repetitions. We expect T_N to be linear with N: test whether indeed $T_N = a+bN$. Interpret a, b.

Explain: how many floating point operations per loop have to be performed in a) and b) ? In variant a), you additionally have to account for the time to access the memory. How do times T_N for a) and b) compare ? What is the work/memory ratio for this algorithm ? (we define the work/memory ratio as the ratio between the number of operations that have to be performed divided by the number of memory locations referenced).

Extra: you could estimate the performance of the computer you are using in terms of FLOPS (floating point operations per second). For variant b), the performance in FLOPS would be the total number of floating point operations that need to be performed, divided by the time in seconds. Try it on this computer and on some other computers you have access to.

3. Newton's algorithm: an M-file as provided by a link from lab page.

Execute the algorithm `newton(2,1e-14)` that is, with initial guess 2, tolerance 1e-14. **Explore:** Use other tolerance and initial guesses. Test the q-convergence rate.

Determine experimentally the interval around the true solution such that if initial guess is included in it, the convergence will be quadratic. **Explain** using information in class (forthcoming) and theory from [Kelley, Chapter 5].

Change the function to $f(x) = x^2 - 2x + 1$ and test convergence to the root. What problems do you anticipate ?

Improve the algorithm: consider the following issues:

- what happens when you call `newton(0,1e-8)` ? how to prevent problems ?
- what to do if the algorithm is not converging ?
- change stopping criteria
- add testing for difference between subsequent iterates

Extra: The last part has an interesting feature in that a good *estimate* for ε_{n+1} is $\varepsilon_{n+1} \approx \upsilon_{n+1} = |x_{n+1} - x_n|$. Test it. Note that this is a unique feature of Newton's algorithm.

~~4. **Extra:** Stability of numerical derivatives:~~

Use the one-sided difference formula $D_h f(x) = \frac{f(x+h)-f(x)}{h}$ to approximate the derivative of $f(x) = \cos(x)$ at $x = .5$, with h ranging from $1E-1$ down to the machine epsilon (step by a factor of 1/10). Discuss behavior of the error. What is the behavior of central derivative ? (You may want to use `loglog` plot).