18.330 Pset 5

Kathleen Brandes

March 21, 2019

1 Problem 1

The incorrect order Richardson's Extrapolation will have order equal to that of the original function and worse than the correct Richardson's Extrapolation. This can be seen by going through the process of formulating Richardson's extrapolation with correct order p and incorrect order q. Deriving the expression for $F_{richardson}(\Delta)$, we get:

$$F(\Delta) = F(0) + C\Delta^p + \mathcal{O}(\Delta^{p+1})$$
$$F(\Delta/2) = F(0) + C(\tfrac{\Delta}{2})^p + \mathcal{O}(\Delta^{p+1})$$

These two expressions have the correct order p because they are exact expressions for the function F, however when we go next in the process of deriving Richardson's Extrapolation to multiplying by $2^{\texttt{order}} = 2^q$ with what we incorrectly believe is the order for the function $F(\Delta)$, we get:

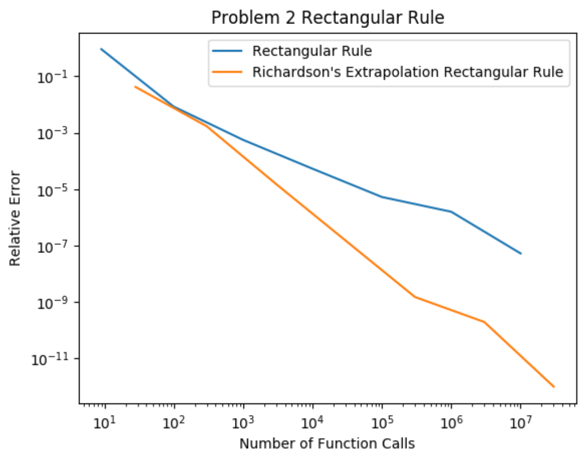
$$\begin{aligned} 2^q F(\Delta) - F(\Delta) &= 2^q F(0) - F(0) + 2^{q-p} C\Delta^p - C\Delta^p + \mathcal{O}(\Delta^{p+1}) \\ &= (2^q - 1)F(0) + C\Delta^p(2^{q-p} - 1) + \mathcal{O}(\Delta^{p+1}) \end{aligned}$$

Normally, at this stage the $C\Delta^p$ terms will cancel out because multiplying $F(\Delta/2)$ by 2^p removes the fractional constant factor. However, since the order was incorrectly estimated as q, this term can no longer be cancelled out. This makes it so that the estimate still has the same error convergence as the initial function, $\mathcal{O}(\Delta^p)$, to a constant factor rather than getting an improvement like Richardson's Extrapolation guarantees.

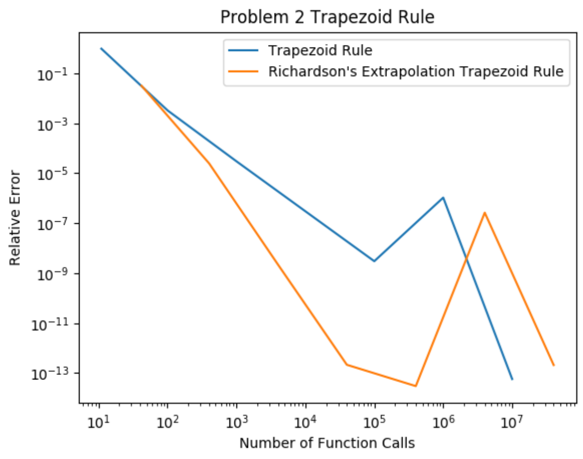
2 Problem 2

The code for this problem can be found in stellar in the file problem2.jl. All problems used a generic Richardson's function which can be passed in the original function, one of the Newton-Cotes rules, an order of the function, an initial delta, and a value t. For all three quadrature rules, I plotted the relative error of the original rule and the Richardson's extrapolated rule versus the number

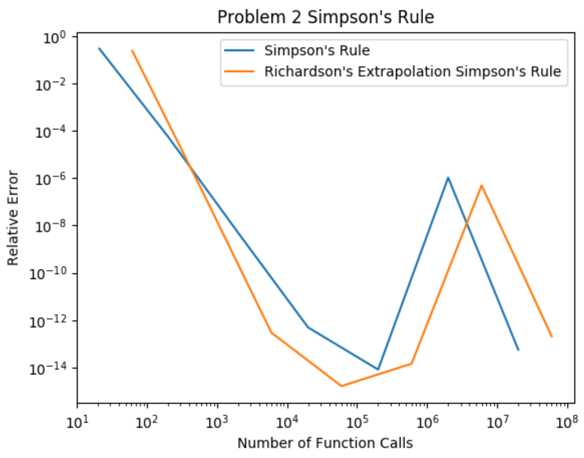
of function calls used.
For the rectangular rule:



For the trapezoid rule:



For Simpson's rule:



For all three quadrature rules, while Richardson's extrapolation does improve the error convergence rate, it is at the cost of more function evaluations

3 Problem 3

3.1 Part A

The polynomial expression for $g(x_n)$ is a forward difference stencil approximation for $f(x_n)$ using a step size $h = f(x_n)$.

3.2 Part B

Let $\epsilon_n = |x - x^*|$ with x^* being the exact root such that $f(x^*) = 0$. First, we can taylor expand the function around the root to get $f(x) = f(x^*) + (x - x^*)f'(x^*) + \mathcal{O}((x - x^*)^2)$. Then we plug this approximation into the expression for g to a taylor-approximation of this function around the root x^*:

$$
\begin{gathered}
g(x) = \frac{f(x+f(x))-f(x)}{f(x)} = \frac{f(x+(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2))-((x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2))}{(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2)} \\
= \frac{(x+(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2)-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2))-((x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2))}{(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2)} \\
= \frac{(x-x^*)f'(x^*)+(x-x^*)(f'(x^*))^2+\mathcal{O}((x-x^*)^2)-((x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2))}{(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2)} \\
= \frac{(x-x^*)(f'(x^*))^2+\mathcal{O}((x-x^*)^2)}{(x-x^*)f'(x^*)+\mathcal{O}((x-x^*)^2)} \\
= f'(x^*) + \mathcal{O}((x - x^*)^2)
\end{gathered}
$$

Then we can express the update:

$$
\begin{gathered}
x_{n+1} = x_n - \frac{f(x)}{g(x)} \\
x_{n+1} - x^* = x_n - x^* - \frac{f(x_n-x^*)}{g(x_n-x^*)} \\
x_{n+1} - x^* = x_n - x^* - \frac{(x_n-x^*)f'(x^*)+\mathcal{O}((x_n-x^*)^2)}{f'(x^*)+\mathcal{O}((x_n-x^*)^2)}
\end{gathered}
$$

Now if we let $\epsilon_n = |x - x^*|$ represent the error between the actual root and the estimated root. We can rewrite the update step of Steffensen's method as such:

$$\begin{aligned}\epsilon_{n+1} &= \epsilon_n - \tfrac{\epsilon_n f'(x^*)+\mathcal{O}(\epsilon_n^2)}{f'(x^*)+\mathcal{O}(\epsilon_n^2)} \\ &= \epsilon_n(1 - \tfrac{f'(x^*)+\mathcal{O}(\epsilon_n)}{f'(x^*)+\mathcal{O}(\epsilon_n^2)}) \\ &= \epsilon_n(1-(1+\mathcal{O}(\epsilon_n))) \\ &= \mathcal{O}(\epsilon_n^2)\end{aligned}$$

Then, to express the error in terms of ϵ_0, we can iterate:

$$\epsilon_{n+1} = \mathcal{O}(\epsilon_n^2) = \mathcal{O}(\mathcal{O}(\epsilon_{n-1}^2)^2) = ... = \mathcal{O}(\epsilon_0^{2^N})$$

This shows that Steffensen's method has quadratic convergence for the error, which is the same convergence rate as Newton's method when ignoring constant terms.

4 Problem 4

My code for problem 4 is attached on stellar in the file problem4.jl. This code solves systems of equations using Newton's method for N dimensions.

4.1 Part A

The roots for this system of equations were: $[x_1, x_2] = [0.5, 0.866025]$, and they had a relative tolerance of $6.149722512997785e - 9$.

4.2 Part B

The roots for this system of equations were: $[x_1, x_2, x_3] = [0.498145, -0.199606, -0.528826]$, with a relative tolerance of $1.2740468650405443e - 11$.

5 Problem 5

My code for this part is attached on stellar under problem5.jl.

5.1 Part A

To find the optimal step size, we want to minimize the expression: $min_{\alpha_k} f(x_k - \alpha_k \nabla f(x_k))$. To do this, first we need to take the gradient of the function f: $\nabla f(x_k) = x^T Q - b^T$. Now, using this gradient, we can solve for the minimum by taking the derivative with respect to α_k and set it equal to zero to solve for an expression:

$$(x_k - \alpha_k \nabla f(x_k))^T Q(-\nabla f(x_k)) - b^T(-\nabla f(x_k)) = 0$$

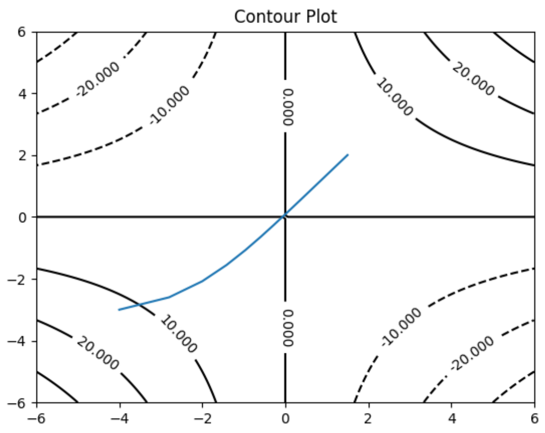
$$\alpha_k = \frac{-b^T(\nabla f(x_k)) + x_k^T Q(\nabla f(x_k))}{(\nabla^T f(x_k))Q(\nabla f(x_k))}$$

$$\alpha_k = \frac{(\nabla^T f(x_k))(\nabla f(x_k))}{(\nabla^T f(x_k))Q(\nabla f(x_k))}$$

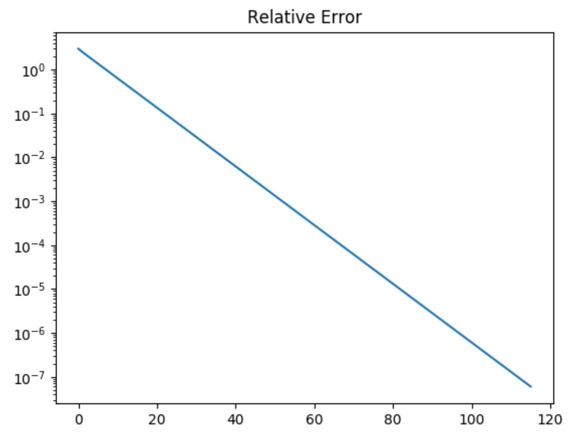
5.2 Part B

For the initial starting point of $x = [-4, -3]^T$, and function $f(x) = \frac{1}{2}x^T Qx - b^T x$, the steepest descent algorithm iterates by updating the vector using $x_{k+1} = x_k - \alpha_k \nabla f(x_k) = [x_1, x_2] - \alpha_k[x_1 Q_{11} + x_2 Q_{21} - b_1, x_1 Q_{12} + x_2 Q_{22} - b_2]$.

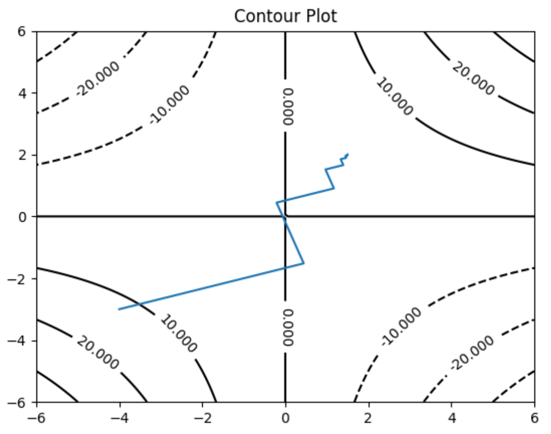
Using a constant $\alpha_k = .1$, the steepest descent algorithm converged to $x = (1.4999999077406496, 1.9999998818364009)$, which evaluated is $f(x^*) = 0$. Here is a contour plot showing the algorithms iteration to converge to this minimum value.



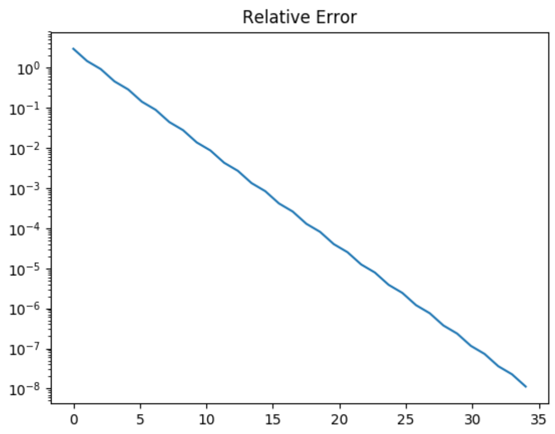
And here is a plot showing the relative error for this calculation, which converges linearly, making this method order-1.



Using the optimal alpha calculated in part a, the steepest descent algorithm converged to $x = (1.4999999919410067, 1.9999999731366884)$, which evaluated is $f(x^*) = 0$. Here is a contour plot showing the algorithms iteration to converge to this minimum value.



And here is a plot showing the relative error for this calculation, which also converges linearly with the optimal alpha, confirming that the steepest descent method is order-1.



5.3 Part C

Applying Newton's method to find roots when $\nabla f(x) = 0$, I found that the algorithm would immediately solve for the exact roots of this gradient function in a single iteration, returning the same answer as part b of $x = [1.5, 2.0]$ from multiple different starting positions. This convergence is very quick, requiring no iterations. This is potentially because the functions are rather simple, linear functions for the gradient and so finding the root is an easy problem to solve with a system of equations.