Report for Project 2

Zhihan Li

1600010653

June 19, 2018

1 Algorithms

The function given is

$$f(x) = \frac{1}{1+\mathrm{e}^{-x}} - \frac{1}{2} = \frac{1}{2}\tanh\frac{1}{2}x, \tag{1}$$

and we have

$$f'(x) = \frac{1}{4\cosh^2\frac{1}{2}x}. \tag{2}$$

The algorithm of Newton method is given in Algorithm 1.

Data: Initial guess $x^{(0)}$, tolerance ϵ, bound ω
$i \leftarrow 0$;
while ***True*** **do**
  if $\left|f\left(x^{(i)}\right)\right| < \epsilon$ **then**
    break;
  end
  if $\left|x^{(i)}\right| > \omega$ **then**
    flag the failure;
    break;
  end
  $x^{(i+1)} = x^{(i)} - f\left(x^{(i)}\right) / f'\left(x^{(i)}\right)$;
  $i \leftarrow i + 1$;
end
return *final result* $x^{(i)}$ *and the number of iterations* i

Algorithm 1: Newton method to find the zero of f

To alleviate stability issues of Newton method, we may use continuation methods to approximate initial guesses. The continuation method involves solving the differential equation

$$\begin{cases} x'(t) = -f\left(x^{(0)}\right)/f'(x(t)); \\ x(0) = x^{(0)}, \end{cases} \tag{3}$$

where $x^{(0)}$ is the original initial guess. The modified algorithm is shown in Algorithm 2.

Data: Initial guess $x^{(0)}$, tolerance ϵ, bound ω
solve the continuation differential equation for $x(1)$;
return *results from Newton method with initial guess* $x(1)$*, and* ϵ*,* ω

Algorithm 2: Continuation method to refine the initial guess

There are different algorithms to solve the differential algorithm. Forward Euler method can be directly carried out by Algorithm 3. Here nodes i/n are denoted by t_i respectively, and

$$\phi(x) = -f\left(x^{(0)}\right)/f'(x) \tag{4}$$

represents local slopes in (3). Additionally, h stands for $1/n$, the step size.

Data: Initial value $x(0)$, number of intervals n
for *i from* 0 *to* $n-1$ **do**
  $x(t_{i+1}) = x(t_i) + h\phi(x(t_i))$;
end
return *final value* $x(1)$

Algorithm 3: Forward Euler method to solve continuation differential equation

We use iterations to perform backward Euler method. This is shown in Algorithm 4. The algorithm using trapezoid method turns out to be similar and is shown in Algorithm 5. The only difference is iteration step.

Runge-Kutta method of order 4 is also implemented. The is depicted in Algorithm 6.

2 Experiments

Question 1

Answer When the initial guess is set to be $x^{(0)}$ = -2.17731898 with ϵ = 1e-3, the approximate solution is $x^{(i)}$ = -1.78753444e-04, and the number of iterations is i = 18.

Data: Initial value $x(0)$, number of intervals n, criterion μ

for *i from* 0 *to* $n-1$ **do**
    $k \leftarrow 0$;
    $x^{(0)}(t_{i+1}) \leftarrow x(t_i) + h\phi(x(t_i))$;
    while ***True*** **do**
        /* Iterate using backward Euler method */
        $x^{(k+1)}(t_{i+1}) \leftarrow x(t_i) + h\phi\left(x^{(k)}(t_{i+1})\right)$;
        if $\left|x^{(k+1)}(t_{i+1}) - x^{(k)}(t_{i+1})\right| < \mu$ **then**
            break;
        end
        $k \leftarrow k+1$;
    end
    $x(t_{i+1}) = x^{(k)}(t_{i+1})$;
end
return *final value* $x(1)$

Algorithm 4: Backward Euler method to solve continuation differential equation

Data: Initial value $x(0)$, number of intervals n, criterion μ

for *i from* 0 *to* $n-1$ **do**
    $k \leftarrow 0$;
    $x^{(0)}(t_{i+1}) \leftarrow x(t_i) + h\phi(x(t_i))$;
    while ***True*** **do**
        /* Iterate using trapezoid formula */
        $x^{(k+1)}(t_{i+1}) \leftarrow x(t_i) + \frac{1}{2}h\left(\phi(x(t_i)) + \phi\left(x^{(k)}(t_{i+1})\right)\right)$;
        if $\left|x^{(k+1)}(t_{i+1}) - x^{(k)}(t_{i+1})\right| < \mu$ **then**
            break;
        end
        $k \leftarrow k+1$;
    end
    $x(t_{i+1}) = x^{(k)}(t_{i+1})$;
end
return *final value* $x(1)$

Algorithm 5: Trapezoid method to solve continuation differential equation

Data: Initial value $x(0)$, number of intervals n, criterion μ

for *i from* 0 *to* $n-1$ **do**

$k \leftarrow 0$;

$K_1 \leftarrow \phi(x(t_i))$;

$K_2 \leftarrow \phi\left(x(t_i) + \frac{1}{2}hK_1\right)$;

$K_3 \leftarrow \phi\left(x(t_i) + \frac{1}{2}hK_2\right)$;

$K_4 \leftarrow \phi(x(t_i) + hK_3)$;

$x(t_{i+1}) \leftarrow x(t_i) + \frac{1}{6}h(K_1 + 2K_2 + 2K_3 + K_4)$;

end

return *final value* $x(1)$

Algorithm 6: Runge-Kutta method of order 4 to solve continuation differential equation

For ϵ = 1e-16, results of continuation method is shown in Table 1. In practice, the tolerance ϵ are all set to be 1e-16. The number of intervals n are set to be 10, which means the step size $h = 1/10$. The criterion μ are selected to be 1e-5 for Algorithm 4 and 5, which rely on iterations to construct corresponding schemes.

Question 2

Answer When the initial guess is set to be $x^{(0)}$ = -4 with ϵ = 1e-3, it can be observed that $x^{(2)}$ = -6.51107241e+09 and divergence occurs.

For ϵ = 1e-16, results of continuation method is also shown in Table 1. Settings of this experiment is identical to that described in Question 1.

Algorithms		$x^{(0)}$ = -2.17731898		$x^{(0)}$ = -4	
		$x^{(i)}$	i	$x^{(i)}$	i
Direct Newton 1		0.00000000e+00	20	-6.51107241e+09*	2*
Continuation 2	Forward 3	0.00000000e+00	3	0.00000000e+00	4
	Backward 4	-6.31088724e-30	3	0.00000000e+00	3
	Trapezoid 5	3.30872245e-23	2	0.00000000e+00	3
	Runge-Kutta 4 6	0.00000000e+00	2	1.82959117e-19	2

Table 1 Numerical results for different algorithms

*The algorithm goes divergent.

We also record the end points $x(1)$ of the continuation differential equation calculated by each algorithm. The result is shown in Table 2.

The settings are described in Question 1.

All codes are implemented in `Problem.ipynb` and `utils.py` in Python.

Algorithms	$x(0)$ = -2.17731898		$x(0)$ = -4	
	$x(1)$	$f(x(1))$	$x(1)$	$f(x(1))$
None*	-2.17731898e+00	-3.981942e-01	-4.00000000e+00	-4.820138e-01
Forward 3	8.82983701e-02	2.206026e-02	6.65728340e-01	1.605460e-01
Backward 4	-7.40656318e-02	-1.850795e-02	-2.11813309e-01	-5.275623e-02
Trapezoid 5	6.94553122e-03	1.736376e-03	1.00664911e-01	2.514500e-02
Runge-Kutta 4 6	1.83784341e-05	4.594609e-06	1.83576885e-02	4.589293e-03

Table 2 End points $x(1)$ of continuation differential equation of different algorithms
*None means $x(1) = x(0)$ virtually. This is exactly the initial value for direct Newton methods.

3 Conclusion

According to numerical results, we can see that Newton method is sensitive to the initial value: a bad initial value may lead to divergence (the case $x^{(0)} = 4$). Additionally, whether a initial value is good is rather complicated to determine. However, once the convergence region is entered, the convergence of Newton method is very fast. It can be seen that less than 20 iterations result in a full precision result. On the contrary, continuation method is not so sensitive to the initial value. As a non-iterative algorithm, results provided by continuation methods do not enjoy high precision (see Table 2). Additionally, Euler methods has better precision than trapezoid method, which in turn is better than the fourth order Runge-Kutta method. Increment in convergence needs smaller step length, and this introduce extra time expense. Combining continuation method and Newton method together, we may approximate the solution first using coarse intervals ($h = 0.1$ is rather coarse with respect to $[0, 1]$), and then refine it in a fast manner. This leads to overall improvement in stability, speed and convergence for real applications.