DS288 (AUG) 3:0 Numerical Methods

Naman Pesricha
namanp@iisc.ac.in
SR - 24115

Homework-2

---

1. Using Newton's method, Secant method, and Modified Newton's method, find the solution of f(x) = 0 for the functions listed. For the Newton methods start with an initial guess of $p_0 = 0$. For the Secant method start with initial guesses (or interval) of $p_0 = 0$ and $p_1 = 1$. Iterate until you reach a relative tolerance of 10^{-6} between successive iterates. Report the root found and the number of iterations needed for each method.
(a) $f(x) = x + e^{-x^2} cosx$.
(b) $f(x) = (x + e^{-x^2} cosx)^2$.
Comment on the observed convergence rates in these cases. Does your results agree with the analysis we did in class ?. [4 points]

Answer:

	Newtons		Secant		Modified Newtons	
	iterations	root	iterations	root	iterations	root
$f(x) = x + e^{-x^2} cosx$	5	-0.58840178	8	-0.58840178	5	-0.58840178
$f(x) = (x + e^{-x^2} cosx)^2$	19	-0.58840144	35	-0.58840136	5	-0.58840178

Table 1: Root values and iterations taken to converge in Newtons. Secant and Modified Newtons methods.

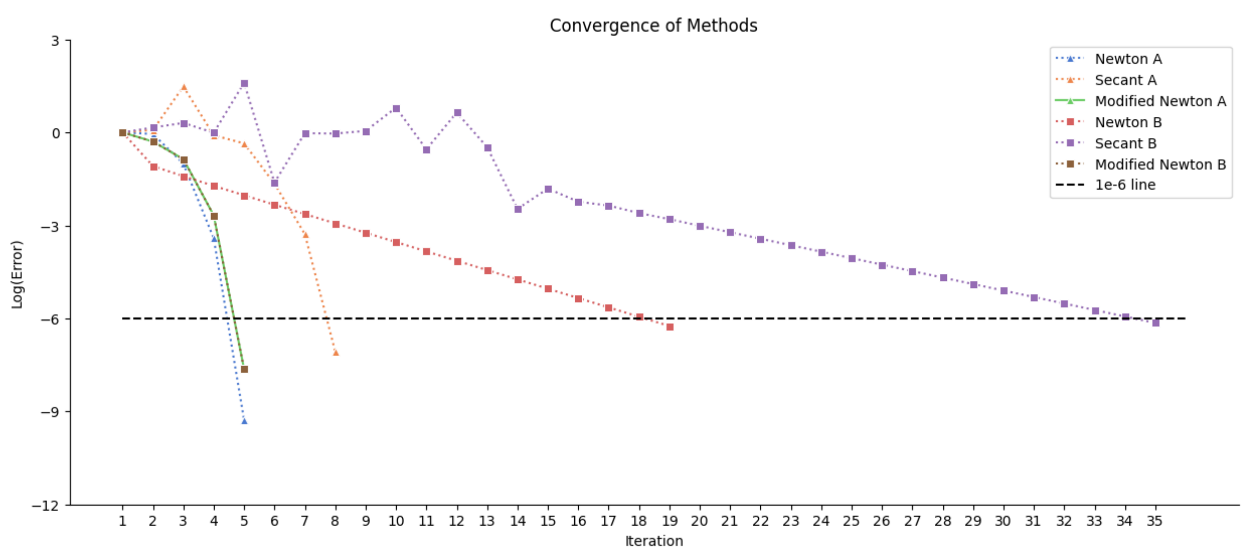

Figure 1: Logarithm of error over the iterations.

Problem a) has a single root at $x = -0.588401$
Problem b) has root at $x = -0.588401$ with multiplicity 2.

From Figure 1, we can make the following observations:
1. Newton's Method:
Problem a): Exhibits Quadratic Convergence.
Reason: Newton's Method has quadratic convergence $\alpha = 2$ for single root problem.

Problem b): Shows Linear convergence trend.
Reason: Newton's Method has linear convergence $\alpha = 1$ if root multiplicity $m \geq 2$.

2. Secant Method:
Problem a): Shows better than Linear convergence but worse than Quadratic Convergence.
Reason: Secant Method has superlinear convergence $\alpha = 1.62$ for single root problem.

Problem b): Shows linear convergence trend.
Reason: Secant Method has linear convergence $\alpha = 1$ if root multiplicity $m \geq 2$.

3. Modified Newton's Method:
Exhibits Quadratic Convergence rate for both problem a) and b).
Reason: Modified Newton's Method exhibits quadratic convergence irrespective of root multiplicity.

4. Newton's Method takes less iterations compared to Secant Method for problem a) and b) as .

For problem a), since $\alpha_{Newton's\ Method} \geq \alpha_{Secant\ Method}$, and hence Newton's converges in less iterations.

For problem b) the asymptotic error constant of Newtons method is less than that of Secant Method and hence Newton's Method converges in less iterations.

$$\lambda_{Newton'sMethod} \ < \ \lambda_{SecantMethod}$$

In general, for same order of convergence α, lower value of $\lambda \implies Faster\ Convergence$.

---

2. Develop the functional form for a cubicly convergent fixed point iteration function $g(p_n)$ to solve the problem f(x) =0 by writing

$$g(x) = x - \phi(x)f(x) - \psi(x)f^2(x)$$

and determining $\phi(x)$ and $\psi(x)$. Specify the asymptotic order of convergence (α) and write the asymptotic error constant (λ). Write all expressions in terms of $f(p)$ and its derivatives and simplify your answers. You are allowed to scan the hand-written derivation for this part alone. Hint: Extend the approach we used in class to derive Newton's method. The scheme you will produce is often referred to as "Cubic Newton's Method". [3 points]

Answer: For cubic convergence, we want $g'(p) = 0$ and $g''(p) = 0$. We also know that $f(p) = 0$ (if p is the root).

We want to evaluate $\phi(x)|f(x) = 0$ and $\psi(x)|f(x) = 0$.

$$\boxed{g(x) = x - \phi(x)f(x) - \psi(x)f^2(x)} \tag{1}$$

$$g'(x) = 1 - \phi'(x)f(x) - \phi(x)f'(x) - \psi'(x)f^2(x) - 2\psi(x)f'(x)f(x)$$
$$(Subsituting\ g'(x) = 0\ and\ f(x) = 0)$$

$$1 - \phi(x) f'(x) = 0$$

$$\textbf{Ans:} \boxed{\implies \phi(x) = \tfrac{1}{f'(x)} \; and \; \phi'(x) = -\tfrac{f''(x)}{(f'(x))^2}} \tag{2}$$

$$We \ will \ now \ find \ g''(x)$$

$$g'' = \phi'' f - \phi' f' - \phi' f' - \phi f'' - \psi'' f^2 - 2\psi' f' f - 2\psi' f' f - 2\psi f'' f - 2\psi (f')^2$$

$$(Subsituting \ g''(x) = 0 \ , \ f(x) = 0)$$

$$-2\phi' f' - \phi f'' - 2\psi (f')^2 = 0$$

$$(Subsituting \ (2))$$

$$2\frac{f''}{(f')^2} f' - \frac{1}{f'(x)} f'' - 2\psi (f')^2 = 0$$

$$\implies 2\psi (f')^2 = 2\frac{f''}{(f')^2} f' - \frac{1}{f'(x)} f''$$

$$\implies 2\psi (f')^2 = \frac{f''}{f'}$$

$$\textbf{Ans:} \boxed{\implies \psi(x) = \tfrac{f''(x)}{2(f'(x))^3}} \tag{3}$$

$$From \ (1) \ , (2) \ and \ (3) \ we \ have$$

$$g(x) = x - \frac{1}{f'(x)} f(x) - \frac{f''(x)}{2(f'(x))^3} f^2(x)$$

$$\textbf{Ans:} \boxed{\implies g(x) = x - \tfrac{f(x)}{f'(x)} [1 + \tfrac{f''(x) f(x)}{2(f'(x))^2}]} \tag{4}$$

From taylor series, ζ between x and p, we know that:

$$g(x) = g(p) + g'(p)(x-p) + \frac{g''(p)}{2}(x - p^2) + \frac{g'''(\zeta)}{6}(x-p)^3$$

$$p_{n+1} = g(p_n) = p + \frac{g'''(\zeta)}{6}(p_n - p)^3$$

$$\frac{p_{n+1} - p}{(p_n - p)^3} = \frac{g'''(\zeta)}{6} \equiv \frac{\epsilon_{n+1}}{\epsilon_n^{\alpha}} = \lambda$$

$$\textbf{Ans:} \boxed{\alpha = 3 \ and \ \lambda = \tfrac{g'''(\zeta)}{6}} \tag{5}$$

3. The figure below shows a four bar linkage where θ_4 is the input angle and the output angles θ_2 and θ_3 are to be determined. The relationships among the linkages can be expressed in terms of the two nonlinear equations

$$f_1(\theta_2, \theta_3) = r_2 cos\theta_2 + r_3 cos\theta_3 + r_4 cos\theta_4 - r_1 = 0$$

$$f_2(\theta_2, \theta_3) = r_2 sin\theta_2 + r_3 sin\theta_3 + r_4 sin\theta_4 = 0$$

Assume $r_1 = 10,\ r_2 = 6,\ r_3 = 8,\ r_4 = 4,\ \theta_4 = 220°$, and solve for θ_2 and θ_3 using Newton's method for systems of nonlinear equations. Compute to a relative tolerance of 10^{-4} and report the number of iterations required to reach this level of convergence. Start with initial guesses of $\theta_2 = 30°$ and $\theta_3 = 0°$. Think about whether θ should be specified in radians or degrees in your code. Invert the 2x2 Jacobian for this problem. [3 points]

Answer:

Iterations	θ_2	θ_3
0	30.00000000	0.00000000
1	32.52053015	-4.70854110
2	32.02031123	-4.37506065
3	32.01518100	-4.37098789
4	**32.01518036**	**-4.37098741**

Table 2: The Scheme Converges in 4th Iteration.

We will use the Newton's Method for system of non linear equations[1].

$$F(\Theta) = \begin{bmatrix} f_1(\theta_2, \theta_3) \\ f_2(\theta_2, \theta_3) \end{bmatrix} \qquad J(\Theta) = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_2} & \frac{\partial f_1}{\partial \theta_3} \\ \frac{\partial f_2}{\partial \theta_2} & \frac{\partial f_2}{\partial \theta_3} \end{bmatrix} \qquad \Theta = \begin{bmatrix} \theta_2 \\ \theta_3 \end{bmatrix}$$

$$G(\Theta) = \Theta - J(\Theta)^{-1} F(\Theta)$$

$$\boxed{\Theta^{(n+1)} = \Theta^{(n)} - J(\Theta^{(n)})^{-1} F(\Theta^{(n)})}$$

Using this iterative scheme, we can find the values of θ_2 and θ_3 (Table 2). The scheme converges in the 4th iteration.

[1] https://math.stackexchange.com/questions/2094608/newton-method-to-solve-nonlinear-system

CODE (Python)

```
# %% [markdown]
# Q1

# %%
import numpy as np
import pandas as pd
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
import sympy as sp

pd.options.display.float_format = '{:.8f}'.format

warnings.filterwarnings("ignore")

x = sp.symbols('x')

f_a = x + sp.exp(-x**2)*sp.cos(x)
f_b = (x + sp.exp(-x**2)*sp.cos(x))**2

d_f_a = sp.diff(f_a, x)
d_f_b = sp.diff(f_b, x)

d2_f_a = sp.diff(d_f_a, x)
d2_f_b = sp.diff(d_f_b, x)

f_a = sp.lambdify(x, f_a)
f_b = sp.lambdify(x, f_b)

d_f_a = sp.lambdify(x, d_f_a)
d_f_b = sp.lambdify(x, d_f_b)

d2_f_a = sp.lambdify(x, d2_f_a)
d2_f_b = sp.lambdify(x, d2_f_b)

def modified_newtons_method(f:callable, df:callable, d2f:callable, p0: np.float64, tol : np.float64 =1e-6 , max_iter: int = 1000) ->tuple:
    p = p0
    p_values = [p]
    error = []
    i = 1
    while i < max_iter:
        p = p - f(p)*df(p)/(df(p)**2 - f(p)*d2f(p))
        p_values.append(p)
        rel_err = abs((p_values[i] - p_values[i-1])/p_values[i])
        error.append(rel_err)
        if rel_err < tol:
            break
        i+=1

    return i,p_values,error

def newton_raphson(f:callable, df:callable,  p0: np.float64, tol : np.float64 =1e-6 , max_iter: int = 1000) ->tuple:
    p = p0
    p_values = [p]
    error = []
    i = 1
    while i < max_iter:
        p = p - f(p)/df(p)
```

```python
        p_values.append(p)
        rel_err = abs((p_values[i] - p_values[i-1])/p_values[i])
        error.append(rel_err)
        if rel_err < tol:
            break
        i+=1

    return i,p_values,error

def secant(f:callable, p0: np.float64, p1: np.float64, tol : np.float64 =1e-6 , max_iter: int = 1000) ->tuple:
    p_values = [p0,p1]
    i = 1
    error = []
    while i < max_iter:
        p = p1 - f(p1)*(p1-p0)/(f(p1)-f(p0))
        p_values.append(p)
        rel_err = abs((p_values[i] - p_values[i-1])/p_values[i])
        error.append(rel_err)
        if rel_err < tol:
            break
        p0 = p1
        p1 = p
        i+=1
    return i,p_values, error

iterations_newton_f_a, p_values_newton_f_a, error_values_newton_a = newton_raphson(f_a, d_f_a, 0)
iterations_newton_f_b, p_values_newton_f_b, error_values_newton_b = newton_raphson(f_b, d_f_b, 0)
iterations_secant_f_a, p_values_secant_f_a, error_values_secant_a = secant(f_a, 0, 1)
iterations_secant_f_b, p_values_secant_f_b, error_values_secant_b= secant(f_b, 0, 1)
iterations_modified_newton_f_a, p_values_modified_newton_f_a, error_values_MN_a = modified_newtons_method(f_a, d_f_a, d2_f_a, 0)
iterations_modified_newton_f_b, p_values_modified_newton_f_b, error_values_MN_b = modified_newtons_method(f_b, d_f_b, d2_f_b, 0)
max_length = max(
    len(p_values_newton_f_a),
    len(p_values_newton_f_b),
    len(p_values_secant_f_a),
    len(p_values_secant_f_b),
    len(p_values_modified_newton_f_a),
    len(p_values_modified_newton_f_b)
    )

p_values_newton_f_a.extend([np.nan] * (max_length - len(p_values_newton_f_a)))
p_values_newton_f_b.extend([np.nan] * (max_length - len(p_values_newton_f_b)))
p_values_secant_f_a.extend([np.nan] * (max_length - len(p_values_secant_f_a)))
p_values_secant_f_b.extend([np.nan] * (max_length - len(p_values_secant_f_b)))
p_values_modified_newton_f_a.extend([np.nan] * (max_length - len(p_values_modified_newton_f_a)))
p_values_modified_newton_f_b.extend([np.nan] * (max_length - len(p_values_modified_newton_f_b)))

data = {
    'Newton a)': p_values_newton_f_a,
    'Secant a)': p_values_secant_f_a,
    'Newton b)': p_values_newton_f_b,
    'Secant b)': p_values_secant_f_b,
    'Modified Newton a)': p_values_modified_newton_f_a,
    'Modified Newton b)': p_values_modified_newton_f_b
}

table = pd.DataFrame(data)
iterations = pd.DataFrame(data =
```

```python
        {
            'Newton' : [iterations_newton_f_a, iterations_newton_f_b],
            'Secant' : [iterations_secant_f_a, iterations_secant_f_b],
            'Modified Newton' : [iterations_modified_newton_f_a, iterations_modified_newton_f_b]
        }
        ,
        index=['a', 'b']
    )

iterations
print(error_values_newton_b)
print(error_values_secant_b)
print(error_values_MN_b)


# %%
table

# %%
plt.figure(figsize=(15, 6))

colors = sns.color_palette("muted", 6)
sns.lineplot(y=np.log10(error_values_newton_a), x = range(1, len(error_values_newton_a)+1), color=colors[0], marker='^', linestyle=':', label='Newton A')
sns.lineplot(y=np.log10(error_values_secant_a), x = range(1, len(error_values_secant_a)+1), color=colors[1], marker='^', linestyle=':', label='Secant A')
sns.lineplot(y=np.log10(error_values_MN_a), x = range(1, len(error_values_MN_a)+1), color=colors[2], marker='^', linestyle='-', label='Modified Newton A')
sns.lineplot(y=np.log10(error_values_newton_b), x = range(1, len(error_values_newton_b)+1), color=colors[3], marker='s', linestyle=':', label='Newton B')
sns.lineplot(y=np.log10(error_values_secant_b),x = range(1, len(error_values_secant_b)+1), color=colors[4], marker='s', linestyle=':', label='Secant B')
sns.lineplot(y=np.log10(error_values_MN_b),x = range(1, len(error_values_MN_b)+1), color=colors[5], marker='s', linestyle=':', label='Modified Newton B')

# max_length = len(error_values_secant_b)
# iterations = np.arange(1,13)
# linear_convergence = -iterations  # Linear convergence line
# iterations = np.arange(1, 5)
# quadratic_convergence = -iterations**2  # Quadratic convergence line

# plt.plot(range(1,5) ,quadratic_convergence, 'k-.', label='Quadratic Convergence')
# plt.plot(range(1,13), linear_convergence, 'k--', label='Linear Convergence')
plt.plot(range(1, 37),[-6]*36, color='black', linestyle='--', label='1e-6 line')
plt.xlabel('Iteration')
plt.ylabel('Log(Error)')
plt.title('Convergence of Methods')
plt.yticks(np.arange(-12, 6, 3))
plt.xticks(np.arange(1, 36, 1))
plt.legend()

plt.gca().spines['top'].set_visible(False)
plt.gca().spines['right'].set_visible(False)


plt.show()

# %% [markdown]
# Q3

# %%
theta2, theta3 = sp.symbols('theta2 theta3')

r1, r2, r3, r4 = 10, 6, 8, 4
```

```python
theta4 = np.deg2rad(220)

f1 = r2 * sp.cos(theta2) + r3 * sp.cos(theta3) + r4 * sp.cos(theta4) - r1
f2 = r2 * sp.sin(theta2) + r3 * sp.sin(theta3) + r4 * sp.sin(theta4)

F = sp.Matrix([f1, f2])

J = F.jacobian([theta2, theta3])


# This is a workaround to convert the SymPy expressions to NumPy functions taken with the
    help of copilot.
F_func = sp.lambdify((theta2, theta3), F, 'numpy')
J_func = sp.lambdify((theta2, theta3), J, 'numpy')
# end of workaround

theta_values = []

def newton_raphson_system(F_func, J_func, theta0, tol=1e-4, max_iter=1000):
    theta = theta0
    theta_values.append([np.rad2deg(theta[0]), np.rad2deg(theta[1])])
    for i in range(1, max_iter + 1):
        F_val = np.array(F_func(theta[0], theta[1]), dtype=np.float64).flatten()
        J_val = np.array(J_func(theta[0], theta[1]), dtype=np.float64)

        delta_theta = np.linalg.solve(J_val, -F_val)
        theta = theta + delta_theta
        theta_values.append([np.rad2deg(theta[0]), np.rad2deg(theta[1])])
        if np.linalg.norm(delta_theta, ord=2) / np.linalg.norm(theta, ord=2)  < tol:
            return i, theta

    return max_iter, theta

theta_initial = np.array([np.deg2rad(30), np.deg2rad(0)], dtype=np.float64)

iterations, theta = newton_raphson_system(F_func, J_func, theta_initial)

# Convert the solution back to degrees
theta2_sol, theta3_sol = np.rad2deg(theta)

print(f"Solution: theta2 = {theta2_sol} degrees, theta3 = {theta3_sol} degrees")
print(f"Number of iterations: {iterations}")

# %%
theta_values = pd.DataFrame(theta_values, columns=['theta2', 'theta3'], index=range(
    iterations + 1))
theta_values

# %%
error_values_MN_a

# %% [markdown]
# ----
```