Newton Method and Fixed Point Iteration

Assignment 1 - Math 4020U

Syed Arham Naqvi (100590852)

February 11, 2025

1 Question 1

(a) The function $f(x) = \cos(x) - x^3 - 2x$ is continuous on $[0, 1]$ since it is the sum of functions $\cos(x)$, $-x^3$ and $-2x$, which are all continuous on $[0, 1]$.

[175]:
```python
def f(x):
    return np.cos(x)-(x**3)-(2*x)
print(f"f(0)={f(0)}")
print(f"f(1)={f(1)}")
```

```
f(0)=1.0
f(1)=-2.4596976941318602
```

Since $f(x)$ is a continuous, $f(0)$ is negative and $f(1)$ is positive, then by IVT there must exist at least one $x \in [0, 1]$ s.t. $f(x) = 0$.

Note also that $\forall x \in \mathbb{R},\ \ f'(x) = -\sin(x) - 3x^2 - 2 < 0 \to f$ is strictly decreasing.

$\therefore$ since $f(x)$ is strictly decreasing and by IVT has at least one root in $[0, 1]$, this root must be unique $\quad\blacksquare$.

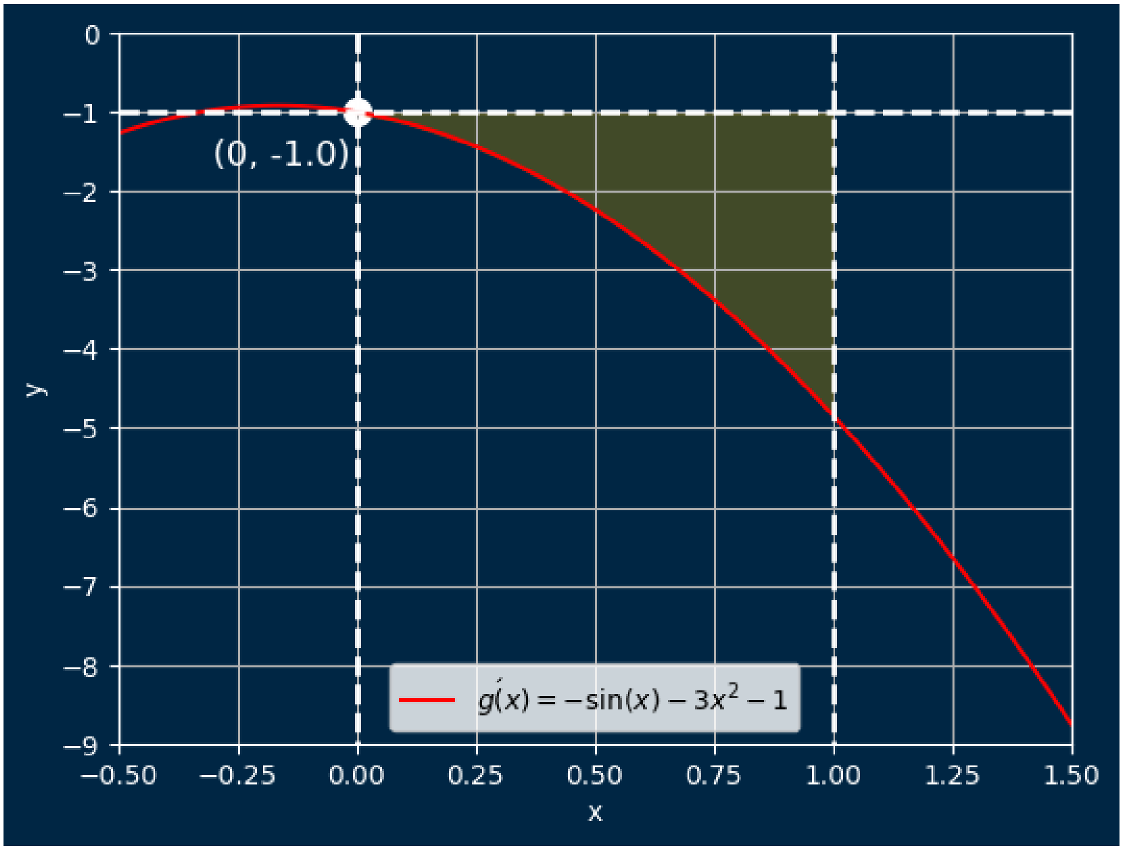
(b) If g has a fixed point x^* where $f(x) = 0$ then by the above result we know $x^* \in [0, 1]$. And if x^* can be approximated by g using fixed point iteration then g is a contraction over an interval $[a, b] \ni x^*$ where $[a, b] \subseteq [0, 1]$. This implies $|g(x^*)| < 1$.

However,

$$\begin{aligned} g'(x) &= f'(x) + x' \\ &= -\sin(x) - 3x^2 - 2 + 1 \\ &= -\sin(x) - 3x^2 - 1 \\ &\leq -1 \quad \forall x \in [0, 1] \end{aligned}$$

and since $x^* \in [0, 1]$ then $|g'(x^*)| \geq 1$ which means there is no interval $[a, b] \ni x^*$ s.t. g is a contraction on $[a, b]$.

$\therefore$ since there exists no interval $[a, b] \ni x^*$ on which g is a contraction, fixed point iteration cannot be used to approximate the fixed point solution. $\quad\blacksquare$



(c) We know $\exists x^* \in [0,1]$ s.t. $f(x^*) = 0$. Thus, we can show

$$\begin{aligned} f(x^*) &= 0 \\ \cos(x^*) - (x^*)^3 - 2(x^*) &= 0 \\ \cos(x^*) &= (x^*)^3 + 2(x^*) \\ \frac{\cos(x^*)}{(x^*)^2 + 2} &= x^* \\ h(x^*) &= x^*, \end{aligned}$$

which proves that the root of f is simultaneously the fixed point $h(x^*) = x^*$.

The reason we can use h to approximate the solution from any initial point in $[0,1]$ is because h is a contraction on $[0,1]$. We show this as follows,

$$\begin{aligned} \frac{d}{dx}h(x) &= \frac{d}{dx}\frac{\cos(x)}{(x^2+2)} \\ h'(x) &= (-1)\left[\frac{\sin(x)(x^2+2) + \cos(x)(2x)}{(x^2+2)^2}\right] \end{aligned}$$

note that $\forall x \in [0,1]$,

$$\begin{aligned} 0 \leq \ \sin(x) \ &< 1 \\ 0 \leq \ \sin(x)(x^2+2) \ &< (x^2+2) \end{aligned}$$

also

$$\begin{aligned} 0 < \ \cos(x) \ &\leq 1 \\ 0 \leq \ \cos(x)(2x) \ &\leq (2x) \end{aligned}$$

which implies

$$\begin{aligned} \cos(x)(2x) + \sin(x)(x^2+2) &< (2x) + (x^2+2) \\ &= (x+1)^2 \\ &< (x^2+2)^2 \end{aligned}$$

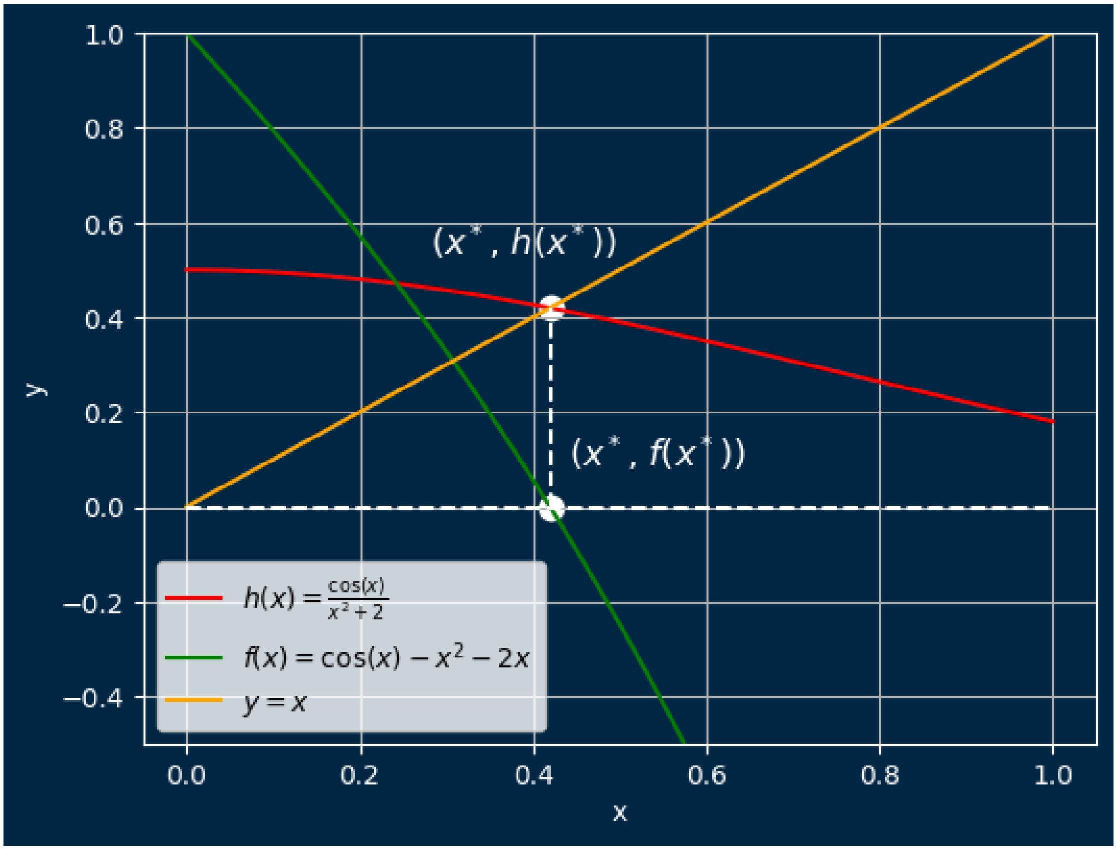
thus we find that $\forall x \in [0,1], \ |h'(x)| < 1$ and $h'(x) \leq 0$.

Finally, since we now know h is non-increasing on $[0,1]$, we conclude $0 < h(1) \approx 0.18 \leq h(x) \leq h(0) = 0.5 < 1$.

[177]:
```
def h(x):
    return np.cos(x)/(x**2 + 2)

display(Markdown(rf'$0 < h(1) \approx {h(1)} \leq h(x) \leq h(0) = 0.5 < 1$'))
```

$0 < h(1) \approx 0.18010076862271326 \leq h(x) \leq h(0) = 0.5 < 1$

$\therefore$ h is a contraction on the interval $[0,1]$ and so fixed point iteration will successfully approximate the solution from any initial point in the interval. ■



(d) We begin by using initial value $x_0 = 0.1$, evaluating the first fixed point iterate $x_1 = h(x_0)$ and estimating λ by sampling from the interval $[0, 1]$ in order to approximate $\max_{x \in [0,1]} |h'(x)|$.

[180]:
```
def h_p(x):
    return (-(x**2+2)*(np.sin(x))-(2*x)*(np.cos(x))) / (x**2+2)**2

lm = np.max(np.abs(h_p(np.linspace(0,1,10000)))) # using 10000 samples in [0,1]
x_0 = 0.1
x_1 = h(x_0)

display(Markdown(rf'$x_0 = {x_0}$'))
display(Markdown(rf'$x_1 = {x_1}$'))
display(Markdown(rf'$\lambda = {lm}$'))
```

$x_0 = 0.1$

$x_1 = 0.4950269478995154$

$\lambda = 0.433149702034376$

We can now approximate the number of iterations required to estimate the fixed point of h within a maximum error threshold of 10^{-2} as follows:

$$
\begin{aligned}
|x^* - x_k| \leq \frac{\lambda^k}{1-\lambda}|x_1 - x_0| \ &\leq \ 10^{-2} \\
\frac{\lambda^k}{1-\lambda}|x_1 - x_0| &\leq 10^{-2} \\
\lambda^k &\leq \frac{(1-\lambda)10^{-2}}{|x_1 - x_0|} \\
k &\geq \frac{\ln\left(\frac{(1-\lambda)10^{-2}}{|x_1-x_0|}\right)}{\ln\lambda} \quad \text{noting that } \ln\lambda < 0
\end{aligned}
$$

[181]:
```python
max_k = (np.log( (1-lm)*(1e-2) / (np.abs(x_1-x_0))) / np.log(lm))
display(Markdown(rf'$k \geq {max_k}$'))
```

$k \geq 5.07251293581352$

$\therefore$ we need at least 6 iterations to compute the fixed point within an absolute error of 10^{-2} $\blacksquare$.

(e) Fixed Point Iteration:

[182]:
```python
x_curr = 0.1
x_next = h(x_curr)
err = np.abs(x_next - x_curr)

for k in range(0,6):
    display(Markdown(rf'$x_{k} = {x_curr}$'))
    display(Markdown(rf'$f(x_{k}) = {f(x_curr)}$'))
    display(Markdown(rf'$x_{k+1} = {x_next}$'))
    display(Markdown(rf'$|x_{k+1} - x_{k}| = {err}$'))
    #updating
    x_curr = x_next
    x_next = h(x_curr)
    err = np.abs(x_next-x_curr)
    print("-------------------")
```

$x_0 = 0.1$

$f(x_0) = 0.7940041652780259$

$x_1 = 0.4950269478995154$

$|x_1 - x_0| = 0.3950269478995154$

$x_1 = 0.4950269478995154$

$f(x_1) = -0.231405172193189$

$x_2 = 0.391953520083751$

$|x_2 - x_1| = 0.1030734278157644$

$x_2 = 0.391953520083751$

$f(x_2) = 0.08004268594944497$

$x_3 = 0.4291199676930137$

$|x_3 - x_2| = 0.03716644760926269$

$x_3 = 0.4291199676930137$

$f(x_3) = -0.027927522408994876$

$x_4 = 0.4163334833353607$

$|x_4 - x_3| = 0.012786484357653005$

$x_4 = 0.4163334833353607$

$f(x_4) = 0.009746324605613754$

$x_5 = 0.42081798803499865$

$|x_5 - x_4| = 0.004484504699637959$

$x_5 = 0.42081798803499865$

$f(x_5) = -0.003402607255166612$

$x_6 = 0.4192550711476302$

$|x_6 - x_5| = 0.0015629168873684263$

(f) We know $f(x) = \cos(x) - x^3 - 2x$. This implies $f'(x) = -\sin(x) - 3x^2 - 2$. Thus, we write the Newton-Raphson update step,

$$x_{k+1} = x_k - \frac{f(x)}{f'(x)}$$
$$x_{k+1} = x_k - \frac{\cos(x_k) - x_k^3 - 2x_k}{-\sin(x_k) - 3x_k^2 - 2}.$$

and implement the algorithm.

[183]:
```
def step(x):
    return x - (np.cos(x)-x**3-2*x) / (-np.sin(x)-3*x**2-2)

x_curr = 1
```

```python
x_next = x_curr
res = 3
k = 0
while (res > 1e-2) :
    #updating
    x_curr = x_next
    x_next = step(x_curr)
    err = np.abs(x_next-x_curr)
    res = np.abs(f(x_curr))

    #printing
    display(Markdown(rf'$x_{k} = {x_curr}$'))
    display(Markdown(rf'$f(x_{k}) = {f(x_curr)}$'))
    display(Markdown(rf'$x_{k+1} = {x_next}$'))
    display(Markdown(rf'$|x_{k+1} - x_{k}| = {err}$'))
    print("-------------------")

    k+=1
    if(k > 100):
        break
```

$x_0 = 1$

$f(x_0) = -2.4596976941318602$

$x_1 = 0.5789249487793613$

$|x_1 - x_0| = 0.4210750512206387$

$x_1 = 0.5789249487793613$

$f(x_1) = -0.5148276453641975$

$x_2 = 0.43400866852358666$

$|x_2 - x_1| = 0.14491628025577463$

$x_2 = 0.43400866852358666$

$f(x_2) = -0.042481385431854535$

$x_3 = 0.419779917047197$

$|x_3 - x_2| = 0.014228751476389634$

$x_3 = 0.419779917047197$

$f(x_3) = -0.0003527678923979094$

$x_4 = 0.4196597728850908$

$|x_4 - x_3| = 0.00012014416210620604$

---

2 Question 2

(a) The Newton-Raphson method for solving the nonlinear system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$, where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$, is given by the iterative update:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J_{\mathbf{f}}(\mathbf{x}_k)^{-1}\mathbf{f}(\mathbf{x}_k)$$

where:

- $\mathbf{x}_k$ is the current estimate of the root.
- $J_{\mathbf{f}}(\mathbf{x}_k)$ is the Jacobian matrix of $\mathbf{f}$ at $\mathbf{x}_k$, defined as:

$$J_{\mathbf{f}}(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

- $J_{\mathbf{f}}(\mathbf{x}_k)^{-1}$ is the inverse of the Jacobian matrix, assuming it is non-singular.

Alternatively, we can avoid the computationally expensive and inaccurate task of finding $J_{\mathbf{f}}(\mathbf{x}_k)^{-1}$, solving directly for $\Delta\mathbf{x}_k$ in the equation:

$$J_{\mathbf{f}}(\mathbf{x}_k)\Delta\mathbf{x}_k = -\mathbf{f}(\mathbf{x}_k)$$

where $\Delta\mathbf{x}_k = \mathbf{x}_{k+1} - \mathbf{x}_k$, and then using $\Delta\mathbf{x}_k$ to update $\mathbf{x}_{k+1}$.

[184]:
```
# calculates and returns jacobian at given x
def J(x_k):
    x = x_k[0]
    y = x_k[1]

    df1dx = (6*x) * (y**2) - 1
    df1dy = (6*x**2) * y

    df2dx = (2*x*y)
    df2dy = (x**2) - 10*y

    return np.array([[df1dx, df1dy],[df2dx, df2dy]])

# vector valued function
def f(x_k):
    x = x_k[0]
```

```python
    y = x_k[1]
    return np.array([
        (3*x**2)*(y**2) - x - 1,
        (x**2)*(y) - 5*(y**2) - 1
    ])

# initial point
x_curr = np.array([2, 0.5])
display(Markdown(rf'$x_{0} = [{x_curr[0]}, {x_curr[1]}]$'))
display(Markdown(rf'$res_{0} = {np.linalg.norm(f(x_curr))}$'))
print("-------------------")

for k in range(0,3):

    J_f = J(x_curr)
    f_x = f(x_curr)
    delta_x = np.linalg.solve(J_f, -f_x)
    x_next = delta_x + x_curr
    x_curr = x_next

    display(Markdown(rf'$x_{k+1} = [{x_next[0]}, {x_next[1]}]$'))
    display(Markdown(rf'$res_{k+1} = {np.linalg.norm(f(x_next))}$'))
    print("-------------------")
```

$x_0 = [2.0, 0.5]$

$res_0 = 0.25$

$x_1 = [2.1153846153846154, 0.4807692307692308]$

$res_1 = 0.013175191940192588$

$x_2 = [2.1176136187276184, 0.481399464873907]$

$res_2 = 2.6707631153480953e-05$

$x_3 = [2.117610262196714, 0.4813979659055765]$

$res_3 = 1.0080634270820037e-10$

(b) Given the equations,

$$f_1(x, y) = 3x^2y^2 - x - 1 = 0$$
$$f_2(x, y) = x^2y - 5y^2 - 1 = 0$$

we can rearrange as follows,

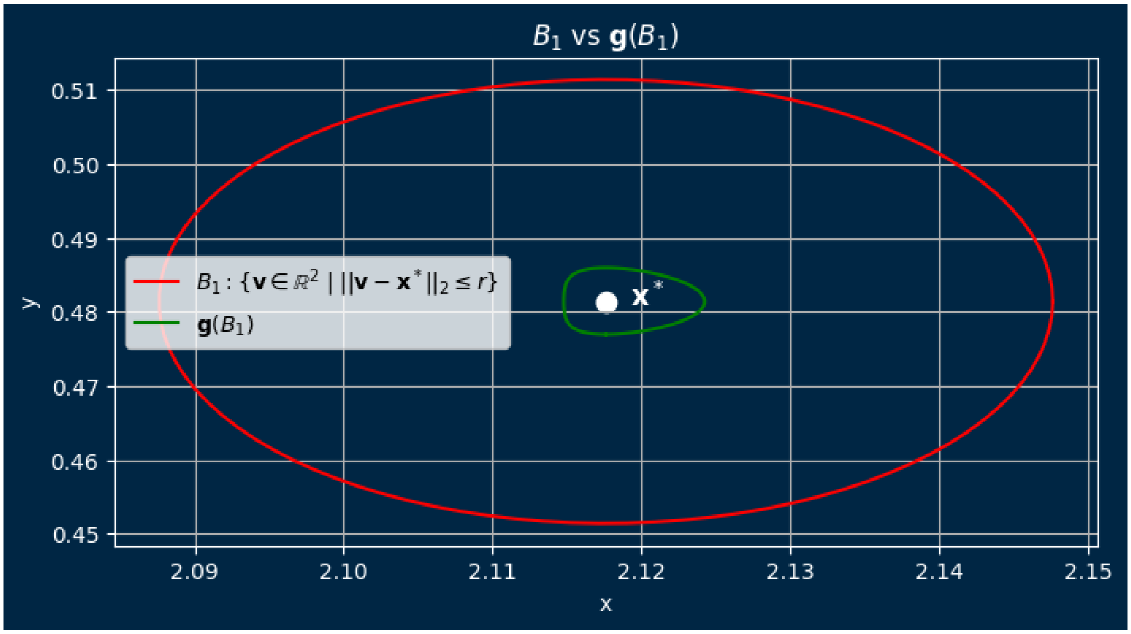
$$g_1(x) = \sqrt{\frac{x+1}{3x^2}} = y$$
$$g_2(y) = \sqrt{\frac{1+5y^2}{y}} = x$$

Allowing us to write the system $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ as,

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} g_2(y) \\ g_1(x) \end{bmatrix} = \begin{bmatrix} \sqrt{\frac{1+5y^2}{y}} \\ \sqrt{\frac{x+1}{3x^2}} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} = \mathbf{x}$$

[185]:
```
r = 0.03
x_fixed = x_curr[0]
y_fixed = x_curr[1]

def g2(y):
    return np.sqrt(((5*y**2) + 1) / y)
y = np.repeat([y_fixed], 100) + r*np.sin(np.linspace(0, 2*np.pi,100))
xs = g2(y)

def g1(x):
    return np.sqrt((x + 1) / (3 * x**2))
x = np.repeat([x_fixed], 100) + r*np.cos(np.linspace(0, 2*np.pi,100))
ys = g1(x)


plt.figure(figsize=(8, 4))
plt.plot(x, y, '-r', label=r'$B_1: \{\mathbf{v}\in\mathbb{R}^2 \; | \;␣
↪||\mathbf{v} - \mathbf{x}^*||_2 \leq r \}$')
plt.plot(xs, ys, '-g', label=r'$\mathbf{g}(B_1)$')
plt.scatter(x=x_fixed, y=y_fixed, c='white', s=80)
plt.annotate(r'$\mathbf{x}^* $ ', (x_fixed, y_fixed), textcoords='offset␣
↪points', xytext=(20,-2), ha='center', color="white", fontsize=13)

fig = plt.gcf()
ax = plt.gca()
fig.set_facecolor('#002644')
ax.set_facecolor('#002644')
for s in ax.spines.values():
    s.set_color('white')
ax.grid()
ax.legend()
ax.set_title(r'$B_1$ vs $\mathbf{g}(B_1)$', color='white')
ax.set_xlabel('x', c='white')
ax.set_ylabel('y', c='white')
```

```
ax.tick_params('both', colors="white")
plt.show()

plt.show()
```



We can confirm through the above graph that $\mathbf{g}$ is a contraction on $B_1 \ni \mathbf{x}^*$ meaning we can use fixed point iteration to converge to $\mathbf{x}^*$ from any $\mathbf{x_0}$ in B_1 $\blacksquare$.

(c) The residual of the third iteration of our Newton-Raphson iterations was on the order of 10^{-10}. The number of iterations required to achieve the same accuracy as three Newton-Raphson iterations can be estimated by solving for k as follows:

We use the fixed point functions from part (b),

$$g_1(x) = \sqrt{\frac{x+1}{3x^2}}, \quad g_2(y) = \sqrt{\frac{1+5y^2}{y}}$$

to evaluate $D\mathbf{g}(\mathbf{x})$ at $\mathbf{x}^* = (2.1176, 0.4814)$,

$$Dg(\mathbf{x}^*) = \begin{bmatrix} 0 & \frac{dg_2}{dy} \\ \frac{dg_1}{dx} & 0 \end{bmatrix}$$

$$\frac{dg_1}{dx} = -\frac{(x+2)}{2\sqrt{3}\,x^2\sqrt{x+1}}$$

$$\frac{dg_2}{dy} = \frac{5y^2-1}{2y^{3/2}\sqrt{1+5y^2}}$$

[192]:
```
def G(x):
    return np.array([g2(x[1]), g1(x[0])])
x_curr = np.array([2,0.5])
```

[187]:
```
x = x_curr[0]
y = x_curr[1]
dg1dx = -(x + 2) / (2*np.sqrt(3)*x**2*np.sqrt(x+1))
dg2dy = (5*y**2 - 1) / (2*y**(3/2) * np.sqrt(1 + 5*y**2))

display(Markdown(rf'$\frac{{dg1}}{{dx}} = {dg1dx}$'))
display(Markdown(rf'$\frac{{dg2}}{{dy}} = {dg2dy}$'))
```

$\frac{dg1}{dx} = -0.16666666666666669$

$\frac{dg2}{dy} = 0.2357022603955158$

Next we find the largest singular value (2-norm) of $Dg(\mathbf{x}^*)$. For this diagonal Jacobian we have:

$$\|Dg(\mathbf{x}^*)\|_2 \approx \max\left\{\left|\frac{dg_1}{dx}\right|, \left|\frac{dg_2}{dy}\right|\right\} = 0.1615$$

Thus, we set $\lambda = \left|\frac{dg_2}{dy}\right|$.

Next, we find $\|\mathbf{x}_1 - \mathbf{x}_0\| = \|\mathbf{g}(\mathbf{x}_0) - \mathbf{x}_0\|$,

[188]:
```
eps=np.linalg.norm(G(x_curr) - x_curr)
eps
```

[188]: np.float64(0.12132034355964239)

We can now solve for k using the second property of contraction and our maximum error threshold,

$$\begin{aligned}
\|\mathbf{x}^* - \mathbf{x}_{k+1}\| \leq \frac{\lambda^k}{1-\lambda}\|\mathbf{x}_1 - \mathbf{x}_0\| &\leq 10^{-10} \\
\frac{\lambda^k}{1-\lambda}\|\mathbf{x}_1 - \mathbf{x}_0\| \leq&\ 10^{-10} \\
\lambda^k \leq&\ \frac{(1-\lambda)10^{-10}}{\|\mathbf{x}_1 - \mathbf{x}_0\|} \\
k \geq&\ \frac{\ln\left(\frac{(1-\lambda)10^{-10}}{\|\mathbf{x}_1 - \mathbf{x}_0\|}\right)}{\ln\lambda} \quad \text{noting that } \ln\lambda < 0
\end{aligned}$$

[189]:
```python
x_0 = x_curr
x_1 = G(x_curr)
lm = np.abs(dg2dy)

k = np.log( (1-lm)*(1e-10) / eps ) / np.log(lm)
k
```

[189]:
```
np.float64(14.659240951932732)
```

$\therefore$ we need k $\geq$ 13 iterations to acheive the same accuracy obtained after three Newton-Raphson iterations $\blacksquare$.

We compute the first three iterates using fixed point iteration:

[191]:
```python
display(Markdown(rf'$x_{k} = [{x_curr[0]}, {x_curr[1]}]$'))
print("--------------------")
x_next = x_curr
for k in range(0,3):
    #updating
    x_next = G(x_next)
    #printing
    display(Markdown(rf'$x_{k+1} = [{x_next[0]}, {x_next[1]}]$'))
    print("--------------------")
```

$x_12 = [2.0, 0.5]$

$x_1 = [2.1213203435596424, 0.5]$

$x_2 = [2.1213203435596424, 0.48084188080672774]$

$x_3 = [2.1175209888926116, 0.48084188080672774]$

----

3 Question 3

Claim. The following statements are equivalent for a function $g \in C^1[a,b]$ and a constant $\lambda < 1$:

$$\begin{aligned} &1. \quad |g(x) - g(y)| \le \lambda\,|x - y| \quad \forall\, x, y \in [a,b],\\ &2. \quad |g'(x)| \le \lambda \quad \forall\, x \in [a,b]. \end{aligned}$$

Proof

(1) $\to$ (2):

Assume $|g(x) - g(y)| \le \lambda\,|x - y|$ for all $x, y \in [a,b]$.

Letting $h = x - y$, we get

$$|g(x+h) - g(x)| \le \lambda\,|h|$$

$$\left|\frac{g(x+h) - g(x)}{h}\right| \le \lambda.$$

Taking the limit as $h \to 0$,

$$\begin{aligned} \lim_{h\to 0}\left|\frac{g(x+h) - g(x)}{h}\right| &\le \lim_{h\to 0} \lambda.\\ \left|\lim_{h\to 0}\frac{g(x+h) - g(x)}{h}\right| &\le \lambda.\\ |g'(x)| &\le \lambda. \end{aligned}$$

(1) $\leftarrow$ (2):

Assume $|g'(x)| \le \lambda$ for all $x \in [a,b]$. Let $x, y \in [a,b]$ with $x \ne y$. By the Mean Value Theorem, there exists c between x and y such that

$$g(y) - g(x) = g'(c)\,(y - x).$$

Thus,

$$|g(y) - g(x)| = |g'(c)|\,|y - x| \le \lambda\,|y - x|.$$

■