MATH 548. Midterm.

Laziz bek Sadullaev.

Problem 1.

Find (a) the roots and (b) their multiplicities of the following functions in the interval $[0, 2]$ using the Newton's method. (c) Comment on your results and discuss any commonalities between the results of (i) & (ii):

(i) $f(x) = 21.12 - 32.4x + 12x^2$

(ii) $h(x) = 2.7951 - 8,954x + 10,56x^2 - 5,4x^3 + x^4$

Solution:

(i)

(1) $f(x) = 21,12 - 32,4x + 12x^2$

(a) Roots:

The Newton's method iteration formula:

$$(2)\quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Before directly using the Newton's method, let me first verify if Newton's method can be used (and how) by the following 3 tests:

Test ①: Avoiding extreme points for initial starting point x_0:

$$(3)\quad f'(0) \neq 0 \quad \text{or} \quad f'(2) \neq 0$$

(4) $f'(x) = -32.4 + 24x \Rightarrow$

(5) $\begin{cases} f'(0) = -32.4 \neq 0 \\ f'(2) = -32.4 + 48 \neq 0 \end{cases} \Rightarrow$ For now both 0 & 2 can be x_0.

(6) Test ②: Pick $x_0 \in \left\{ |g'(x)| < 1 : g(x) = x - \frac{f(x)}{f'(x)} \right\} \Rightarrow$

(7) $g'(x) = \left(x - \frac{f(x)}{f'(x)}\right)' = 1 - \left(\frac{f(x)}{f'(x)}\right)' = 1 - \frac{f' \cdot f' - f \cdot f''}{[f']^2} =$

$= \frac{[f']^2 - f \cdot f'' - [f']^2}{[f']^2} \Rightarrow$

(7) $g'(x) = \frac{f(x) \cdot f''(x)}{[f'(x)]^2}$

Continuing (4):

(8) $f''(x) = (-32.4 + 24x)' = 24.$ So, $g'(x)$ is

(9) $g'(x) = \frac{(21.12 - 32.4x + 12x^2) \cdot 24}{(-32.4 + 24x)^2}$

Using this (9), now let me check which boundary point (0 or 2) satisfy $|g'(x)| < 1$ to be an initial point x_0:

(10) $|g'(0)| = \left| \frac{21.12 \cdot 24}{(-32.4)^2} \right| = 0.48 < 1 \Rightarrow x_0 = 0$ is ok.

(11) $|g'(2)| = \left| \frac{(21.12 - 32.4 \cdot 2 + 12.2^2) \cdot 24}{(-32.4 + 24.2)^2} \right| = \left| \frac{4.32 \cdot 24}{243.4} \right| = 0.42 < 1 \Rightarrow$

$x_0 = 2$ is also ok.

Test ③: Picking faster convergence point if ∃ more than 1 candidate initial points for x_0:

(12) $|g'(0)| = 0.48 < 0.42 = |g'(2)| \Rightarrow$

$x_0 = 2$ would guarantee faster convergence. But later in my code, to be even faster I'll start the right side boundary of solution intervals. So now we are ready to find the roots of (1) using (2) with $x_0 = 2$:

(1) $f(x) = 21.12 - 32.4x + 12x^2$

- Since we're considering 2nd order equation $f(x) = 0$, there's a possibility that both roots of (1) are in $[0, 2]$. So, here in my code, I also try to find all intervals which have a root, then from there I'll give the roots only in $[0, 2]$ at the end using (2).

[code ↓]

pr1_i_math548_midterm_Lazizbek

March 16, 2024

1 Math 548, Midterm. Problem 1. LS

Problem 1

(20 points) Find the roots and their multiplicities of the following functions in the interval [0,2] using the Newton's method. Comment on your results and discuss any commonalities between the results of (i) and (ii).

i. f(x) = 21.12 - 32.4x + 12x^2

ii. h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4

#In which interval (i) has a solution? Determine all of the solution intervals using the sign comparison of the function at boundaries of the intervals [-b, a] or [c, b] starting symmetrically around both sides of the origin, 0, with a tolerance b as 10^(-3)

[6]:
```
# In which interval it has a solution?

import numpy as np
import math

# f(x) = 21.12 - 32.4*(x) + 12*(x)**2
f0 =  21.12 - 32.4*(0) + 12*(0)**2 # f0 = 21.12
frightb = f0
fleftb = f0
fa = f0
fc = f0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  frightb = 21.12 - 32.4*(b) + 12*(b)**2
  fleftb = 21.12 - 32.4*(-b) + 12*(-b)**2

  if np.sign(fleftb) * np.sign(fa) <= 0: # fleftb <= 0 as fa = f0 = 21.12 > 0:
    print(" = b ", b, ";", "fleftb = ", fleftb)
    solution_ints.append([-b, a])
```

```
    a = -b
    fa = fleftb

  if np.sign(frightb) * np.sign(fc) <= 0: # frightb <= 0 as fc = f0 = 21.12 > 0:
    print("b = ", b, ";", "frightb = ", frightb)
    solution_ints.append([c, b])
    c = b
    fc = frightb

print("solution intervals = ", solution_ints)
```

```
b =  1.101 ; frightb =  -0.005987999999996774
b =  1.6 ; frightb =  3.552713678800501e-15
solution intervals =  [[0, 1.101], [1.101, 1.6]]
```

solution intervals = [] means, I have only multiple roots for my square function because if the given function has any negative values my code would capture it with tolerance of b = 10^(-3), and insert it into solution_ints list as a solution interval.

#Determine the solutions using the Newton's method with a tolerance as 10^(-6)

[2]:
```
# Determine this solution using the Newton's method with a tolerance  as␣
↪10^(-6)
from scipy.optimize import fsolve


M = 10**6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(len(solution_ints)):
  solution_int = solution_ints[i]

  print(f"\nRoot {i+1} of f(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] # by the result of (12), x0=2 would give faster␣
↪convergence
  v = 21.12 - 32.4*(x0) + 12*(x0)**2  # starting function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "f(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = (-32.4 + 24*(x1))**2
    mdenom = mnumer - (21.12 - 32.4*(x1) + 12*(x1)**2)*(24)+0.00000001 # in␣
↪case x0 is a desired root
    m = round(mnumer/mdenom)
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {m}")
```

```
    else:
      for k in range(1, M):
        x1 = x0 - v/(-32.4 + 24*(x0))
        v = 21.12 - 32.4*(x1) + 12*(x1)**2
        print(k, "\t ", x1, "\t", v)
        if abs(v) < epsilon:
          print(f"\nSo the approximate solution {i+1} after {k} steps is = {x1}")
          # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
          # but the most important two is the function you want to find the root,␣
↪and the initial guess interval.
          f = lambda x: 21.12 - 32.4*(x) + 12*(x)**2
          mnumer = (-32.4 + 24*(x1))**2
          mdenom = mnumer - (21.12 - 32.4*(x1) + 12*(x1)**2)*(24)+0.00000001 # in␣
↪case x0 is a desired root
          m = mnumer/mdenom

          print("Using Python's fsolve function, fsolve(f, [0, 2]) = ", ␣
↪*fsolve(f, solution_int[i]), '\n')
          print("m = ", m)
          print(f"multipilicity of the root {x1} is {round(m)}")
          break
        x0 = x1
```

```
Root 1 of f(x) in [0, 1.101] is as follows:
steps     x                    f(x)
1         1.099997991967872    1.2048241156747963e-05
2         1.099999999991936    4.838796030526282e-11

So the approximate solution 1 after 2 steps is = 1.099999999991936
Using Python's fsolve function, fsolve(f, [0, 2]) =  1.0999999999999992

m =  0.9999999997544808
multipilicity of the root 1.099999999991936 is 1

Root 2 of f(x) in [1.101, 1.6] is as follows:
steps     x                    f(x)
0         1.6    3.552713678800501e-15
m =  1
multipilicity of the root 1.6 is 1
```

#In case I calculated the roots of (i) by hand:

[5]:
```
print(32.4**2-4*21.12*12)
print((32.4+6)/(2*12))
```

```
print((32.4-6)/(2*12))
```

```
36.0
1.5999999999999999
1.0999999999999999
```

[↑ code]

(14) $x_1^* = 1.09999$

$x_2^* = 1.6.$ is the result from code.

(B) Multiplicities:

(15) $g'(x^*) = 1 - \frac{1}{m} \Rightarrow$

(16) $m = \text{round}\left[\frac{1}{1-g'(x^*)}\right] = \text{round}\left(\frac{1}{1-\frac{f(x^*)\cdot f''(x^*)}{[f'(x^*)]^2}}\right) \Rightarrow$

(17) $m = \text{round}\left(\frac{[f'(x^*)]^2}{[f'(x^*)]^2 - f(x^*)\cdot f''(x^*)}\right)$, Using (1), (4), (8),

(18) $m_1 = \text{round}\left(\frac{[f'(1.099)]^2}{[f'(1.099)]^2 - f(1.099)\cdot f''(1.099)}\right) = \boxed{1 = m_1}$

Similarly

(19) $m_2 = \text{round}\left(\frac{[f'(1.6)]^2}{[f'(1.6)]^2 - f(1.6)\cdot f''(1.6)}\right) =$

$= \text{round}\left(\frac{[-32.4 + 24\cdot 1.6]^2}{[-32.4+24\cdot 1.6]^2 - [(21.12 - 32.4\cdot 1.6 + 12\cdot (1.6)^2)\cdot 24]}\right) =$

$= \text{round}(0.999) = 1. \Rightarrow \boxed{m_2 = 1}$ (20)

- Since all roots of (1) are in $[0,2]$ by the result from 1st part of the code, all roots in (14) have been given with their multiplicities.

(ii)

(21) $h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4$

(a) Roots;

(22) $x_{n+1} = x_n - \dfrac{h(x_n)}{h'(x_n)}$

similar way, let me check 3 tests for the method;

Test ①: Avoiding extreme points for initial point x_0:

(23) $h'(0) \neq 0$ or $h'(2) \neq 0$.

(24) $h'(x) = -8.954 + 10.56(2x) - 5.4(3x^2) + 4x^3 \Rightarrow$

(25) $\begin{cases} h'(0) = -8.954 \neq 0 \\ h'(2) = 0.486 \neq 0 \end{cases} \Rightarrow$ for now bot 0 & 2 are OK.

Test ②: Pick x_0 s.t.

(26) $x_0 \in \{ |g'(x)| < 1 ; g(x) = x - \dfrac{h(x)}{h'(x)} \}$

By the result from (7):

(27) $g'(x) = \dfrac{h(x) \cdot h''(x)}{[h'(x)]^2}$, continuing (24):

(28) $h''(x) = 10.56 \cdot 2 - 5.4 \cdot (6x) + 12x^2$ ⟶ so $g'(x)$ is

(29) $g'(x) = \dfrac{\overbrace{(2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4)}^{h(x)} \cdot \overbrace{(21.12 - 32.4x + 12x^2)}^{h''(x)}}{\underbrace{(-8.954 + 21.12x - 16.2x^2 + 4x^3)^2}_{h'(x)}}$

- If we pay attention $h''(x) = 21.12 - 32.4x + 12x^2 = f(x)$ in the previous problem (i), so, we might expect similar roots here as well.

Using this (29), let me check which boundary point (0 & 2) satisfy $|g'(x)| < 1$ to be an convergent initial point x_0:

(30) $|g'(0)| = \left| \dfrac{(2.7951) \cdot (21.72)}{(-8.954)} \right| = 9.91 \not< 1 \Rightarrow$

$x_0 = 0$ can not be possible.

(31) $|g'(2)| = \left| \dfrac{(-0.0729) \cdot (4.32)}{0.486} \right| = 0.648 < 1 \Rightarrow$

$x_0 = 2$ is ok to be an initial starting point.

Test③: Picking faster convergence point, if ∃ more than 1 candidates. Here $x_0 = 2$ is the only convergent initial point to start with. Ready!

(21) $h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4$, $x_0 = 2$

[code ↓]

pr1_ii_math548_midterm_Lazizbek

March 17, 2024

1 Math 548, Midterm. Problem 1. LS

Problem 1

(20 points) Find the roots and their multiplicities of the following functions in the interval [0,2] using the Newton's method. Comment on your results and discuss any commonalities between the results of (i) and (ii).

i. f(x) = 21.12 - 32.4x + 12x^2

ii. h(x) = 2.7951 - 8.954x + 10.56x^2 - 5.4x^3 + x^4

#In which interval (ii) has a solution? Determine all of the solution intervals using the sign comparison of the function at boundaries of the intervals [-b, a] or [c, b] starting symmetrically around both sides of the origin, 0, with a tolerance b as 10^(-3)

[3]:
```
# In which interval it has a solution?

import numpy as np
import math

# h(x) = 2.7951 - 8.954*x + 10.56*x**2 - 5.4*x**3 + x**4
h0 = 2.7951 - 8.954*(0) + 10.56*(0)**2 - 5.4*(0)**3 + (0)**4 # h(0) = 2.7951
hrightb = h0
hleftb = h0
ha = h0
hc = h0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  hrightb = 2.7951 - 8.954*(b) + 10.56*(b)**2 - 5.4*(b)**3 + (b)**4
  hleftb = 2.7951 - 8.954*(-b) + 10.56*(-b)**2 - 5.4*(-b)**3 + (-b)**4

  if np.sign(hleftb) * np.sign(ha) <= 0: # hleftb <= 0 as ha = h0 = 2.7951 > 0:
    print(" = b ", b, ";", "hleftb = ", hleftb)
    solution_ints.append([-b, a])
```

```
    a = -b
    ha = hleftb

  if np.sign(hrightb) * np.sign(hc) <= 0: # hrightb <= 0 as ha = h0 = 2.7951 >
 ↪0:
    print("b = ", b, ";", "hrightb = ", hrightb)
    solution_ints.append([c, b])
    c = b
    hc = hrightb

print("solution intervals = ", solution_ints)
```

```
b =  1.1 ; hrightb =  -6.661338147750939e-16
solution intervals =  [[0, 1.1]]
```

solution intervals = [ ], empty list, means that I have only multiple roots for my function because if the given function has any negative values my code would capture it with tolerance of b = 10^(-3), and insert it into solution_ints list as a solution interval.

#Determine the solutions using the Newton's method with a tolerance as 10^(-6)

[4]:
```
# Determine this solution using the Newton's method with a tolerance  as
 ↪10^(-6)
from scipy.optimize import fsolve


M = 10**6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(len(solution_ints)):
  solution_int = solution_ints[i]

  print(f"\nRoot {i+1} of h(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] # by the result of (32), x0=2 would give faster
 ↪convergence even though I am not necessarily starting with x0 = 2
  v = 2.7951 - 8.954*(x0) + 10.56*(x0)**2 - 5.4*(x0)**3 + (x0)**4  # starting
 ↪function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "h(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
    mnumer = ( - 8.954 + 21.12*(x1) - 16.2*(x1)**2 + 4*(x1)**3)**2
    mdenom = mnumer - (2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 +
 ↪(x1)**4)*(21.12 - 32.4*(x1) + 12*(x1)**2)+0.000001 # in case x0 is a desired
 ↪root
    print("mnumer = ", mnumer)
```

```
    print("mdenom = ", mdenom)
    m = mnumer/mdenom
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {round(m)}")

    # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
    # but the most important two is the function you want to find the root, and␣
↪the initial guess interval.
    f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
    print("To check our work, the solution of h(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')

  else:
    for k in range(1, M):
      x1 = x0 - v/( - 8.954 + 21.12*(x0) - 16.2*(x0)**2 + 4*(x0)**3)
      v = 2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 + (x1)**4
      print(k, "\t ", x1, "\t", v)
      if abs(v) < epsilon:
        print(f"\nSo the approximate solution {i+1} after {k} steps is = {x1}")

        mnumer = ( - 8.954 + 21.12*(x1) - 16.2*(x1)**2 + 4*(x1)**3)**2
        mdenom = mnumer - (2.7951 - 8.954*(x1) + 10.56*(x1)**2 - 5.4*(x1)**3 +␣
↪(x1)**4)*(21.12 - 32.4*(x1) + 12*(x1)**2)+0.000001 # in case x1 is the␣
↪actual(real) root
        print("mnumer = ", mnumer)
        print("mdenom = ", mdenom)
        m = mnumer/mdenom
        print("m = ", m)
        print(f"multipilicity of the root {x1} is {round(m)}")

        # The function fsolve takes in many arguments that you can find in the␣
↪documentation,
        # but the most important two is the function you want to find the root,␣
↪and the initial guess interval.
        f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
        print("To check our work, the solution of h(x) = 0 using Python's␣
↪fsolve function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')

        break
      x0 = x1
```

```
Root 1 of h(x) in [0, 1.1] is as follows:
steps     x                      h(x)
0         1.1     -6.661338147750939e-16
mnumer =  0.0
```

```
mdenom =  1e-06
m =  0.0
multipilicity of the root 1.1 is 0
To check our work, the solution of h(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999932258418257
```

2 Here how I checked the roots of h(x), h'(x), h''(x), h''' '(x) to see what's going on with m:

[5]:
```
# h(x) = 0
f = lambda x: 2.7951 - 8.954*(x) + 10.56*(x)**2 - 5.4*(x)**3 + (x)**4
print("To check our work, the solution of h(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, 1000), '\n')
```

```
To check our work, the solution of h(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  2.100000000000002
```

[16]:
```
# h'(x) = 0
f = lambda x: - 8.954 + 21.12*(x) - 16.2*(x)**2 + 4*(x)**3
print("To check our work, the solution of h'(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h'(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999999815114856
```

[17]:
```
# h''(x) = 0
f = lambda x: 21.12 - 32.4*(x) + 12*(x)**2
print("To check our work, the solution of h''(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h''(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.0999999999999992
```

[18]:
```
# h'''(x) = 0
f = lambda x: - 32.4 + 24*(x)
print("To check our work, the solution of h'''(x) = 0 using Python's fsolve␣
↪function, fsolve(f, [0, 2]) = ",  *fsolve(f, solution_int[i]), '\n')
```

```
To check our work, the solution of h'''(x) = 0 using Python's fsolve function,
fsolve(f, [0, 2]) =  1.35
```

[15]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[14]:
```
# !jupyter nbconvert --to pdf /content/pr1_ii_math548_midterm_Lazizbek.ipynb
```

[code ↑]

Again since $h_4(x) = 0$, I've checked all solution intervals possibly in $[0,2]$.

(c) Based on the result from code, I got $m = 0$ for the root $x_1^* = 1.1$. I was suspicious & tried to find other roots of $h(x)$. It has a root $x_2^* = 2.1 \notin [0,2]$. So no need to discuss it. Now then, I also noticed this x_1^* is a root for $h''(x) = f(x) = 0$ from previous part (i).

So, algebraically

$$\begin{cases} h(x_1 = 1.1) = 0 \\ h''(x_1 = 1.1) = 0 \end{cases} \xrightarrow{\text{forces}} h'(x_1) = 0$$

I've verified by software too, above!

because $\begin{cases} h(x_1) = 0 \Rightarrow (x - x_1)\, p_3(x) = 0 \\ h''(x_1) = 0 \Rightarrow (x - x_1) \cdot q_1(x) = 0 \end{cases} \Rightarrow$

$p_3(x) \mid (x - x_1)$ must also work. So

$$\begin{cases} h(x_1) = (x - x_1)\, p_3(x) \\ h'(x_1) = (x - x_1)\, p_2(x) \\ h''(x_1) = (x - x_1)\, p_1(x) \\ h'''(x_1) \neq 0 \end{cases} \Rightarrow h(x) = (x - x_1)^3 \cdot q(x)$$

$$am(x_1) = 3. \quad \blacksquare$$

Problem 2.

Solve the system of linear equations with Jacobi iteration using the initial guess as $x_0 = [0, 0, 0]^T$.

In each case (a) will the Jacobi iteration converge? give justifications for your answer. (b) If yes, find the solutions.

(1) (i) $\underbrace{\begin{bmatrix} 2 & 1 & 6 \\ 8 & 3 & 2 \\ 1 & 5 & 1 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 9 \\ 13 \\ 7 \end{bmatrix}}_{b}$ $\quad$ (ii) $\begin{bmatrix} 8 & 3 & 2 \\ 1 & 5 & 1 \\ 2 & 1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 13 \\ 7 \\ 9 \end{bmatrix}$

$A \cdot x = b$

Solution:

(i)

(a) There are 2 tests to check if Jacobi iteration converges or not:

Test ①: A is strictly diagonally dominant $\Rightarrow$ Jacobi iteration converges

Test ②: $\rho(B_J) < 1 \Rightarrow$ Jacobi iter-n converges;

(2) $\rho(B_J) = \max\{|\lambda| : \lambda \in \sigma(B_J)\}$, $B_J = -D^{-1}(L+U)$

• Let me now check if our matrix A satisfies either of these tests ① or ②?!

Test ①: (Strictly diagonally dominance of A)

$A = [a_{ij}]$, A is strictly diagonally dominant if

(3) $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| \quad \forall\ i = 1, 2 \dots n.$

For our matrix equ-n (1):

(4) $|2| \not> |1| + |6| \Rightarrow$ A is NOT strictly diag-y dominant $\Rightarrow$ There's no guarantee for convergence of Jacobi iteration, it might or might not converge.

Test ① fails.

Test ②: $(\rho(B_J) < 1)$

Recall by (2), $\rho(B_J) = \max\{|\lambda| : \lambda \in \sigma(B_J)\}$

where $B_J = -D^{-1}(L+U)$ where D, L, U are diagonal, lower, upper matrix parts of A respectively.

(5) $A = \begin{bmatrix} 2 & 1 & 6 \\ 8 & 3 & 2 \\ 1 & 5 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 8 & 0 & 0 \\ 1 & 5 & 0 \end{bmatrix}}_{L} + \underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{D} + \underbrace{\begin{bmatrix} 0 & 1 & 6 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}}_{U} \Rightarrow$

(6) $B_J = -\underbrace{\begin{bmatrix} 2 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1}}_{-D^{-1}} \cdot \left(\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 8 & 0 & 0 \\ 1 & 5 & 0 \end{bmatrix}}_{L} + \underbrace{\begin{bmatrix} 0 & 1 & 6 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix}}_{U} \right) = \begin{bmatrix} 0 & -1/2 & -3 \\ -8/3 & 0 & -2/3 \\ -1 & -5 & 0 \end{bmatrix}$

pr2_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 2. LS

Problem 2

Solve the following systems of linear equations with the Jacobi iteration method using the initial guess as [0, 0, 0].

- In each case, will the Jacobi iteration converge to a solution? Give the justification for your answer?
- If yes, find the solutions.

i.

```
[ 2  1  6][x1]   [ 9 ]

[ 8  3  2][x2] = [ 13]

[ 1  5  1][x3]   [ 7 ]
```

ii.

```
[ 8  3  2][x1]   [ 13]

[ 1  5  1][x2] = [ 7 ]

[ 2  1  6][x3]   [ 9 ]
```

2 Finding eigenstuff of a matrix

Source:

```
[ ]: import numpy as np
     from numpy.linalg import eig
     a = np.array([[0, 2],
                   [2, 3]])
     w,v=eig(a)
     print('E-value:', w)
     print('E-vector', v)
```

```
E-value: [-1.  4.]
E-vector [[-0.89442719 -0.4472136 ]
```

```
[ 0.4472136  -0.89442719]]
```

3 Solving Ax=b matrix equation

Source:

#Problem 2. (i)

To check with the actual(real) solution, here, I'm giving the real solution as well:

```
[ ]: # To solve Ax=b, We start by constructing the arrays for A and b.

A = np.array([[ 2,  1,  6],
              [ 8,  3,  2],
              [ 1,  5,  1]])
b = np.transpose(np.array([ 9,  13,  7]))
# To solve the system we do

x = np.linalg.solve(A,b)
print("Real solution: ")
print(x)
```

```
Real solution:
[1. 1. 1.]
```

Using Jacobi Iteration:

```
[ ]: A = np.array([[ 2,  1,  6],
              [ 8,  3,  2],
              [ 1,  5,  1]])

L = np.array([[ 0,  0,  0],
              [ 8,  0,  0],
              [ 1,  5,  0]])

D = np.array([[ 2,  0,  0],
              [ 0,  3,  0],
              [ 0,  0,  1]])

U = np.array([[ 0,  1,  6],
              [ 0,  0,  2],
              [ 0,  0,  0]])

D_inverse =  np. linalg. inv(D)
b = np.transpose(np.array([ 9,  13,  7]))
D_inverse_b = np.dot(D_inverse, b)

BJ = np.dot(-D_inverse, L+U)
w,v=eig(BJ)
```

```
print('BJ evalues:', w)

# x0 = np.transpose(np.array([ 0,  0,  0]))
# x = list();
# x.append(x0)

# for i in range(10):
#   x1 = np.dot(BJ, x0)+ D_inverse_b
#   x0 = x1
#   x.append(x1)
# Aproximations = np.array(x)
# print(Aproximations)
# print((2.083**2 + 2.312**2)**(1/2))
```

```
BJ evalues: [-4.16531114+0.j          2.08265557+2.31207612j
2.08265557-2.31207612j]
```

Because the spectral radius of matrix BJ, P(BJ) = 4.16531114 > 1, Jacobi Iteration does NOT converge, so no need to turn on the part of the code for Jacobi iteration carried out.

[1]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

(7) $\sigma(B_J) = \{-4.163,\ 2.08 + 2.31i,\ 2.08 - 2.31i\} \Rightarrow$

(8) $\rho(B_J) = \max\{|\lambda| : \lambda \in \sigma(B_J)\} = 4.1653 \not< 1 \Rightarrow$

$\Rightarrow$ Jacobi iteration does NOT converge!

(6) Since (i) fails in both Test ① & ②, no need to see the iterations.

- solution vectors in my code are <u>row vectors</u>!

(ii)
$$\begin{bmatrix} 8 & 3 & 2 \\ 1 & 5 & 1 \\ 2 & 1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 13 \\ 7 \\ 9 \end{bmatrix}$$
(11)
$\quad\quad A \qquad\quad x \qquad\quad b$

- Now if we pay attention this problem (ii) is some appropriately altered (flipped rows) version of problem (i), where we have strictly diagonally dominance after row flips.

Similar steps like before, I 2 tests to check if jacobi iter-n converges or not:

(12)
- Test ①: A is strictly diagonally dominant $\Rightarrow$ Jacobi iteration converges
- Test ②: $\rho(B_J) < 1 \Rightarrow$ Jacobi iteration converges

Let me now check if our A in (11) satisfies any of these Tests ① & ②:

Test①: (Strictly diagonal dominance of A):

(13) $A = [a_{ij}]$, $|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|$ $\forall i = 1, 2, \ldots n.$

For our matrix equation (11):

(14)
$|8| > |3| + |2|$ ✓
$|5| > |1| + |1|$ ✓
$|6| > |2| + |1|$ ✓

$\Rightarrow$ A is strictly diag-ly dominant $\Rightarrow$ Jacobi iteration converges.

(6) A satisfies Test①, so we can start the Jacobi iteration. No need for checking Test②. But during the code, I'll also provide Test② is also satisfied for more detail:

[code ↓]

pr2_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 2. LS

Problem 2

Solve the following systems of linear equations with the Jacobi iteration method using the initial guess as [0, 0, 0].

- In each case, will the Jacobi iteration converge to a solution? Give the justification for your answer?
- If yes, find the solutions.

ii.

```
[ 8  3  2][x1]   [ 13]

[ 1  5  1][x2] = [ 7 ]

[ 2  1  6][x3]   [ 9 ]
```

2 Finding eigenstuff of a matrix

Source:

```
[ ]: import numpy as np
     from numpy.linalg import eig
     a = np.array([[0, 2],
                   [2, 3]])
     w,v=eig(a)
     print('E-value:', w)
     print('E-vector', v)
```

```
E-value: [-1.  4.]
E-vector [[-0.89442719 -0.4472136 ]
 [ 0.4472136  -0.89442719]]
```

3 Solving Ax=b matrix equation

Source:

#Problem 2. (ii)

To check with the actual(real) solution, here, I'm giving the real solution as well:

```
[ ]: # To solve Ax=b, We start by constructing the arrays for A and b.

     A = np.array([[ 8,  3,  2],
                   [ 1,  5,  1],
                   [ 2,  1,  6]])
     b = np.transpose(np.array([ 13,  7,  9]))
     # To solve the system we do

     x = np.linalg.solve(A,b)
     print("Real solution: ")
     print(x)
```

```
Real solution:
[1. 1. 1.]
```

Using Jacobi Iteration:

```
[ ]: A = np.array([[ 8,  3,  2],
                   [ 1,  5,  1],
                   [ 2,  1,  6]])

     L = np.array([[ 0,  0,  0],
                   [ 1,  0,  0],
                   [ 1,  1,  0]])

     D = np.array([[ 8,  0,  0],
                   [ 0,  5,  0],
                   [ 0,  0,  6]])

     U = np.array([[ 0,  3,  2],
                   [ 0,  0,  1],
                   [ 0,  0,  0]])

     D_inverse =  np. linalg. inv(D)
     b = np.transpose(np.array([ 13,  7,  9]))
     D_inverse_b = np.dot(D_inverse, b)

     BJ = np.dot(-D_inverse, L+U)
     w,v=eig(BJ)

     print('BJ evalues:', w)

     x0 = np.transpose(np.array([ 0,  0,  0]))
     x = list();
     x.append(x0)
```

```
for i in range(5):
  x1 = np.dot(BJ, x0)+ D_inverse_b
  x0 = x1
  x.append(x1)
Aproximations = np.array(x)
print(Aproximations)
# print((2.083**2 + 2.312**2)**(1/2))
```

```
BJ evalues: [-0.44378452  0.26976158  0.17402294]
[[0.          0.          0.        ]
 [1.625       1.4         1.5       ]
 [0.725       0.775       0.99583333]
 [1.08541667 1.05583333 1.25      ]
 [0.9165625   0.93291667 1.143125  ]
 [0.989375    0.9880625   1.19175347]]
```

Also pay attention that the spectral radius of matrix BJ, $P(BJ) = 0.44378452 < 1$, Jacobi Iteration does converge.

```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

Problem 3

Use fixed-point iteration to find the roots of

(0) $f(x) = x - x^2 = 0.$

Consider the following 2 formulations:

(i)

(1) $x = x + 2 \cdot (x - x^2)$

(ii)

(2) $x = x - \dfrac{x - x^2}{1 - 2x}$

(a) For each formulation carry out the iterations first using the starting value $x_{01} = 0.8$ & then using the starting value $x_{02} = 0.2$.

(b) Comment and justify your observations:

Solution:

(i)

(1) $x = x + 2(x - x^2)$

(a) In the code, 10 iterations have been carried out on (1) first using $x_{01} = 0.8$, then $x_{02} = 0.2$. I'll attach the code below:

[code ↓]

pr3_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 3. LS

Problem 3

You would like to use the fixed-point iteration method to find the roots of f (x) = x - x^2 = 0. Consider the following two formulations.

1.

`x = x + 2(x-x^2).`

2.

`x = x - (x - x^2) / (1 - 2x).`

a) For each formulation carry out the iterations first using the starting value 0.8 and then, using the starting value 0.2.

b) Comment and justify your observations.

#Problem 3. (i)

x = x + 2*(x-x^2).

[1]:
```
import pandas as pd

initials = list([0.8, 0.2])
steps = list()
approximations = list()
epsilon = 0.000001

for i in range(2):
  x0 = initials[i]
  M = 10
  try:
    for k in range(M):
      steps.append(k)
      approximations.append(x0)
      x1 = x0 + 2*(x0-(x0)**2)
      if abs(x0-x1) < epsilon:
        print(f"\nWhen x0={initials[i]}, |g'({initials[i]})| < 1, so iteration␣
↪converges with tolerance of {epsilon} in {k} steps as follows:")
```

```
            break
        x0 = x1


    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []

  except:
    print(f"\nWhen x0={initials[i]}, |g'({initials[i]})|>=1, so iteration␣
↪diverges in {k} steps as follows:")
    steps.pop()
    approximations.pop()
    d = {'step k = ': steps, 'approximation x = ': approximations}
    df = pd.DataFrame(data=d)
    print(df)
    steps = []
    approximations = []
```

```
   step k =   approximation x =
0         0           0.800000
1         1           1.120000
2         2           0.851200
3         3           1.104517
4         4           0.873635
5         5           1.094429
6         6           0.887738
7         7           1.087057
8         8           0.897786
9         9           1.081319
   step k =   approximation x =
0         0           0.200000
1         1           0.520000
2         2           1.019200
3         3           0.980063
4         4           1.019142
5         5           0.980125
6         6           1.019085
7         7           0.980186
8         8           1.019028
9         9           0.980247
```

```
[ ]: # !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
    ↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

[code ↑]

(6) We can see from the results of (a), the iterations are jumping around 1 with initial value $x_{02} = 0.2$. The reason for this

(1) $x = x + 2(x - x^2) \Rightarrow x = g(x) \Rightarrow$

(4) $g(x) = x + 2 \cdot (x - x^2)$, we know that for x_n

(5) $x_{n+1} = x_n + 2(x_n - x_n^2)$ fixed point iteration to be convergent

(6) $|g'(x)| < 1$ where $g(x) = x + 2(x - x^2)$

If we solve this (6), we find the convergent interval for x_0 initial value of (5):

(7) $|g'(x)| = |1 + 2(1 - 2x)| = |3 - 4x| < 1 \Leftrightarrow$

$-1 < 3 - 4x < 1 \Leftrightarrow \frac{1}{2} < x < 1 \Rightarrow$

(8) $x \in (\frac{1}{2}, 1) =$ convergent interval for (5). $\Rightarrow$

So, if we start $x_0 \in (0.5, 1)$, then (5) converges, otherwise diverges, also

$|g'(x_0)|$ can tell us how fast/slow (5) converges or diverges:

(9) $\begin{cases} \text{Let } x_0 = 0.8 \in (0.5, 1), \text{ so} \\ x_{n+1} = x_n + 2\cdot(x_n - x_n^2) \text{ with } x_0 = 0.8 \end{cases}$

converges as

(10) $|g'(x_0)| = |g'(0.8)| = |3 - 4\cdot 0.8| = 0.2 < 1 \Rightarrow$

(5) converges as in the code.

(11) $\begin{cases} \text{Let } x_0 = 0.2 \notin (0.5, 1), \text{ so} \\ x_{n+1} = x_n + 2(x_n - x_n^2) \text{ with } x_0 = 0.2 \end{cases}$

diverges as

(12) $|g'(x_0)| = |g'(0.2)| = |3 - 4\cdot 0.2| = 2.2 > 1 \Rightarrow$

(5) diverges as in the code.

- Rate of convergence | divergence is determined by:

(13) $\begin{cases} g'(0.8) = 0.2 < 1 \Rightarrow (5) \text{ converges, no other converg. points} \\ g'(0.2) = 2.2 \not< 1 \Rightarrow (5) \text{ diverges, no other divergent points to compare.} \end{cases}$

[code Pr3 ii ↓]

pr3_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 3. LS

Problem 3

You would like to use the fixed-point iteration method to find the roots of f (x) = x - x^2 = 0. Consider the following two formulations.

1.

`x = x + 2(x-x^2).`

2.

`x = x - (x - x^2) / (1 - 2x).`

a) For each formulation carry out the iterations first using the starting value 0.8 and then, using the starting value 0.2.

b) Comment and justify your observations.

2 Problem 3. (ii)

x = x - (x - x^2) / (1 - 2x)

[ ]:
```
import pandas as pd

initials = list([0.8, 0.2])
steps = list()
approximations = list()
epsilon = 0.000000001

for i in range(2):
  x0 = initials[i]
  M = 10
  try:
    for k in range(M):
      steps.append(k)
      approximations.append(x0)
      x1 = x0 - (x0 - (x0)**2) / (1 - 2*x0)
      if abs(x0-x1) < epsilon:
```

```
            print(f"\nWhen x0={initials[i]}, |g'({initials[i]})| < 1, so iteration converges with tolerance of {epsilon} in {k} steps as follows:")
            break
        x0 = x1


      d = {'step k = ': steps, 'approximation x = ': approximations}
      df = pd.DataFrame(data=d)
      print(df)
      steps = []
      approximations = []

    except:
      print(f"\nWhen x0={initials[i]}, |g'({initials[i]})|>=1, so iteration diverges in {k} steps as follows:")
      steps.pop()
      approximations.pop()
      d = {'step k = ': steps, 'approximation x = ': approximations}
      df = pd.DataFrame(data=d)
      print(df)
      steps = []
      approximations = []
```

```
When x0=0.8, |g'(0.8)| < 1, so iteration converges with tolerance of 1e-09 in 4 steps as follows:
   step k =   approximation x =
0          0            0.800000
1          1            1.066667
2          2            1.003922
3          3            1.000015
4          4            1.000000

When x0=0.2, |g'(0.2)| < 1, so iteration converges with tolerance of 1e-09 in 4 steps as follows:
   step k =   approximation x =
0          0         2.000000e-01
1          1        -6.666667e-02
2          2        -3.921569e-03
3          3        -1.525902e-05
4          4        -2.328306e-10
```

```
[ ]: # !sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```

```
[ ]: # !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

[code ↑ pr 3.(ii)]

(a) In the code above, 10 iterations have been carried out on (2) first using $x_{01} = 0.8$ then using $x_{02} = 0.2$.

(b). We can see from the result of (a), the iterations for

$$(2)\quad x = x - \frac{x - x^2}{1 - 2x} \quad \Rightarrow \quad x = g(x),$$

are converging to 1 with both initial value $x_{01} = 0.8$ & $x_{02} = 0.2$. The reason for this

$$(16)\quad g(x) = x - \frac{x - x^2}{1 - 2x}, \quad \text{for } x_n$$

$$(17)\quad x_{n+1} = x_n - \frac{x_n - x_n^2}{1 - 2x_n} \quad \text{iteration}$$

to be convergent we must have x_0, s.t.

$$(18)\quad |g'(x)| < 1$$

If we solve this (18), we find the convergent interval for x_0 initial value x_0 of (17):

(19) $|g'(x)| = \left| 1 - \frac{(1-2x)(1-2x) - (-2)\cdot(x-x^2)}{(1-2x)^2} \right| < 1 \Leftrightarrow$

(20) $\left| \frac{(1-2x)^2 - (1-2x)^2 - 2(x-x^2)}{(1-2x)^2} \right| = \left| \frac{-2(x-x^2)}{(1-2x)^2} \right| < 1$

(21) $\Leftrightarrow \quad -1 < \frac{2x^2 - 2x}{(2x-1)^2} < 1 \quad \Leftrightarrow$

(22) $x \in (-\infty, 0.21) \cup (0.79, \infty) =$

= convergent interval for (17).

So if we start $x_0 \in (-\infty, 0.21) \cup (0.79, \infty)$, then (17) converges, otherwise diverges. Also

$|g'(x_0)|$ can tell us how fast/slow (17) converges or diverges:

(23) [Let $x_{01} = 0.8 \in (-\infty, 0.21) \cup (0.79, \infty)$, so

$x_{n+1} = x_n - \frac{x_n - x_n^2}{1-2x_n}$ with $x_0 = 0.8$ converges

as

(24) $|g'(x_{01})| = |g'(0.8)| = \left| \frac{-2\cdot(0.8-0.8^2)}{(1-2\cdot 0.8)^2} \right| = \frac{8}{9} < 1 \Rightarrow$

$\Rightarrow$ (17) converges

(25) $x_{0_2} = 0.2$

$$|g'(x_{0_2})| = |g'(0,2)| = \left|\frac{-2(0,2 - 0,2^2)}{(1-2\cdot 0,2)^2}\right| = \frac{8}{9} < 1 \Rightarrow$$

(17) converges.

- Rate of convergence:

(26) $$|g'(0,8)| = \frac{8}{9} = \frac{8}{9} = |g'(0,2)| < 1 \Rightarrow$$

$x_{0_1} = 0.8$ & $x_{0_2} = 0,2$ has the same rate of convergence!

Problem 4.

(i) (a) Use Newton's method for finding the root of

$$f(x) = e^x - 2\cos(x) = 0$$

in the interval $[0, 2]$, with starting value $x_0 = 2$. Carry out 6 iterations and comment on what you observe. (b) If the observations need any improvement how will you go about it. (c) Show that the Newton's method's convergence is of second order.

(ii). (d) Now use the iteration method given below to find the root of

$$f(x) = e^x - 2\cos(x) = 0$$

in the interval $[0, 2]$ with starting values $x_0 = 0.6$ & $y_0 = 0.3388$.

$$y_{n+1} = y_n(2 - f'(x_n) \cdot y_n)$$
$$x_{n+1} = x_n - y_{n+1} \cdot f(x_n)$$

(e) Compare your methods & results in (i) & (ii) & discuss any connections between (i) & (ii).

[code ↓]

pr4_i_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 4. LS

Problem 4

(i)

(a) Use the Newton's method for finding the root of

( ) = ^ − 2 cos( ) = 0

in the interval [0,2] with the starting value x0 = 2.

(b) Carry out 6 iterations and comment on what you observe. If the observation needs any improvement, how will you go about it?

(c) Show that the Newton's method's convergence is of second order.

(ii)

(iii) Now, use the iteration method given below to find the root of

( ) = ^ − 2 cos( ) = 0 in the interval [0,2].

Choose your starting values as x0 = 0.6 and y0 = 0.3388.

yn+1 = yn (2 − f'(xn) yn)

xn+1 = xn − yn+1 f(xn)

(e) Compare your methods and results in (i) and (ii) and discuss any connections between (i) and (ii).

#Problem 4. (i)

[ ]:
```
# In which interval it has a solution?

import numpy as np
import math

# f(x) = math.exp(x) - 2*math.cos(x)

f0 =  math.exp(0) - 2*math.cos(0) # f0 = - 1
frightb = f0
fleftb = f0
fa = f0
```

```
fc = f0
solution_ints = list()
b = 0
a = 0
c = 0
# we'll run through [-b, a] or [c, b]

for b in np.arange(0.0, 2.0, 0.001):
  frightb = math.exp(b) - 2*math.cos(b)
  fleftb = math.exp(-b) - 2*math.cos(-b)

  if np.sign(fleftb) * np.sign(fa) <= 0: # fleftb >= 0 as fa = f0 = -1  < 0:
    # print(" = b ", b, ";", "fleftb = ", fleftb)
    solution_ints.append([-b, a])
    a = -b
    fa = fleftb

  if np.sign(frightb) * np.sign(fc) <= 0: # frightb >= 0 as fc = f0 = -1 < 0:
    # print("b = ", b, ";", "frightb = ", frightb)
    solution_ints.append([c, b])
    c = b
    fc = frightb

print("solution intervals = ", solution_ints)
```

```
solution intervals =  [[0, 0.54], [-1.454, 0]]
```

```
[ ]: # Determine this solution using the Newton's method with a tolerance  as␣
     ↪10^(-6)
from scipy.optimize import fsolve


M = 6 # in case the program goes into infinite loops
epsilon = 10**(-8)

for i in range(1):
  solution_int = [0, 2]

  print(f"\nRoot {i+1} of f(x) in {solution_int} is as follows: ")
  x0 = solution_int[1] #  x0=2 as required
  v = math.exp(x0) - 2*math.cos(x0)  # starting function value at x0.

  print("steps", "\t ", "x", "\t", "\t", "\t", "f(x)")

  if abs(v) < epsilon:
    print(0, "\t ", x0, "\t", v)
    x1 = x0
```

```
    mnumer = (math.exp(x1) + 2*math.sin(x1))**2
    mdenom = mnumer - (math.exp(x1) - 2*math.cos(x1))*(math.exp(x1) + 2*math.
↪cos(x1))+0.00000001 # in case x0 is a desired root
    m = mnumer/mdenom
    print("m = ", m)
    print(f"multipilicity of the root {x1} is {round(m)}")

  else:
    for k in range(1, M):
      x1 = x0 - v/(math.exp(x0) + 2*math.sin(x0))
      v = math.exp(x1) - 2*math.cos(x1)
      print(k, "\t ", x1, "\t", v)
      if abs(v) < epsilon:
        mnumer = (math.exp(x1) + 2*math.sin(x1))**2
        mdenom = mnumer - (math.exp(x1) - 2*math.cos(x1))*(math.exp(x1) +␣
↪2*math.cos(x1))+0.00000001 # in case x0 is a desired root
        m = mnumer/mdenom
        print("\nm = ", m)
        print(f"multipilicity of the root {x1} is {round(m)}")
        break
      x0 = x1
```

```
Root 1 of f(x) in [0, 2] is as follows:
steps     x                     f(x)
1         1.1071175683826828    2.1311417447814685
2         0.664462389989102     0.3689486075980428
3         0.5483209267201882    0.023543300836922132
4         0.5398302921852223    0.00012382330442339828
5         0.5397851620829291    3.494306399787206e-09

m =  1.0000000002643719
multipilicity of the root 0.5397851620829291 is 1
```

(c) Show that the Newton's method's convergence is of second order.

For this we need to show $g(x^*) = 0$;

$g(x^*) = 1\text{-}1/m = 1\text{-}1/1 = 0.$

```
[ ]: # !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

```
[ ]: # !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

[code ↑]

Solution: (i)

(a)

(1) $f(x) = e^x - 2\cos(x) = 0$ with $x_0 = 2$

(2) $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{e^{x_n} - 2\cos(x_n)}{e^{x_n} + 2\sin(x_n)}$

(3) $f'(x_0) = f'(2) = e^2 + 2\sin(2) \neq 0$, so no problems to go.

- To make sure if I do not have more than 1 roots of $f(x)$ in $[0,2]$, I tried to see all solution intervals above in the 1st part. Based on that, $\exists$ one (or multiple of this one root only) in interval $[0, 0.54] \subset [0,2]$. Now let me show the Newton's method in $[0,2]$, above.

(6) $x^* = 0.54$ with multiplicity $\boxed{m = 1.}$

I observed that no matter how many iterations (loop, M = 6, 10, 10000 ...) I carried out for testing, the iterations are

always converging in 5 steps with tolerance of $\varepsilon = 10^{-8}$. As for improvements, I'm thinking of the implicit (upgraded) Newton's method, maybe.

(c). Convergence of Newton's method:

From the result of exploring Newton's method convergence in class, we have

$$(7)\quad e_{n+1} = e_n \cdot g'(x^*) + \frac{e_n^2}{2} g''(x^*) + \frac{e_n^3}{6} g'''(x^*) + \dots$$

$$(8)\quad \text{if } g'(x^*) = 0 \overset{(7)\,ii)}{\Longrightarrow} e_{n+1} = \frac{e_n^2}{2} g''(x^*) + \frac{e_n^3}{6} g'''(x^*) + \dots \Rightarrow$$

$\Rightarrow$ Newton's method is 2^{nd} order

$$(9)\quad \text{if } g'(x^*) = g''(x^*) = 0 \overset{(7)\,ii)}{\Rightarrow} e_{n+1} = \frac{e_n^3}{6} g'''(x^*) + \dots \Rightarrow$$

$\Rightarrow$ Newton's method is 3^{rd} order

...

(10) So, to show in our problem Newton's method is 2^{nd} order, by (8) we need to show $\boxed{g'(x^*) = 0}$

where $g(x)$ is $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

$$g(x) = x - \frac{f(x)}{f'(x)}.$$

in part (b) & at the end of code results, we showed that multiplicity m of $x^+ = 0.54$ is 1.

(11) $\boxed{m = 1}$ & we know $\boxed{g'(x^+) = 1 - \frac{1}{m}}$

Plugging this m value into this equation would give the desired result:

(12) $g'(x^+) = 1 - \frac{1}{m} = 1 - \frac{1}{1} = 0 \Rightarrow$ by (8)

Newton's method is 2^{nd} order. ▨

(ii)

(13) $y_{n+1} = y_n \cdot (2 - f'(x_n) \cdot y_n)$

(14) $x_{n+1} = x_n - y_{n+1} \cdot f(x_n)$

(d) I'll attach my code below:

[code ↓ P4.ii]

[code ↑]

(e) Let me discuss what's going on in (d) below. This method gave instantaneous result, so fast, in 3 steps the same tolerance $\varepsilon = 10^{(-8)}$.

If I plug (13) into (14);

$$
\begin{aligned}
x_{n+1} &= x_n - y_n \cdot (2 - f'(x_n) \cdot y_n) \cdot f(x_n) = \\
&= x_n - \frac{f(x_n)}{\dfrac{1}{y_n (2 - f'(x_n) \cdot y_n)}} = \\
&= x_n - \frac{f(x_n)}{\dfrac{1}{2 y_n \left(1 - \frac{1}{2} f'(x_n) \cdot y_n\right)}} = \\
&= x_n - \frac{f(x_n)}{\frac{1}{2} \cdot \left[y_n \cdot \left(1 - \frac{1}{2} f'(x_n) \cdot y_n \right) \right]} \approx
\end{aligned}
\tag{15}
$$

trying to force into

$$
x_n - \frac{f(x_n)}{\frac{1}{2}\left[f'(x_n) + f'(x_{n+1}) \right]}
$$

← Implicit Newton's Method.

- I did a lot of attempt to make this (15) look like something I know (maybe, like Implicit Newton's method) to relate our explicite Newton's method in (i) to (ii) to establish connections, but I couldn't go until the end. ▨

pr4_ii_math548_midterm_Lazizbek

March 19, 2024

1 Math 548, Midterm. Problem 4. LS

Problem 4

(ii)

(iii) Now, use the iteration method given below to find the root of

( ) = ^ − 2 cos( ) = 0 in the interval [0,2].

Choose your starting values as x0 = 0.6 and y0 = 0.3388.

yn+1 = yn (2 – f'(xn) yn)

xn+1 = xn – yn+1 f(xn)

(c) Compare your methods and results in (i) and (ii) and discuss any connections between (i) and (ii).

#Problem 4. (ii)

[ ]:
```
# Determine this solution using the Newton's method with a tolerance  as␣
↪10^(-6)
import numpy as np
import math


M = 6 # in case the program goes into infinite loops
epsilon = 10**(-8)

# f(x) = math.exp(x) - 2*math.cos(x)
x0 = 0.6
y0 = 0.3388
for i in range(1, M):
  y1 = y0*(2 - (math.exp(x0) + 2*math.sin(x0))*y0)
  x1 = x0 - y1*(math.exp(x0) - 2*math.cos(x0))

  v = math.exp(x1) - 2*math.cos(x1)
  print(i, "\t ", x1, "\t", v)
  if abs(v) < epsilon:
    break
  y0 = y1
```

```
    x0 = x1
```

```
1          0.5419098217166501      0.005836841512263824
2          0.5397977761841749      3.461107957947185e-05
3          0.5397851615702651      2.087791273197581e-09
```

[ ]:
```
# !sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
# !jupyter nbconvert --to pdf /content/Math548_hw6_Lazizbek.ipynb
```

Problem 5 (Bonus Pr. 1)

Given

(1) $Ax = b$ where

(2) $A = \text{triadiag}(-1, 2, -1)$ &

(3) $A = 100 \times 100$.

WTS

We need to find (a) $\rho(B_J) = ?$

(b) B_J converges or not? How fast?

Solution:

Let A be

$$(2)\quad A = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{bmatrix}_{100 \times 100}$$

$$(4)\quad B_J = -D^{-1} \cdot (L+U) = \begin{bmatrix} -2 & & & \\ & -2 & & \\ & & \ddots & \\ & & & -2 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0 & -1 & 0 & \dots & 0 & 0 \\ -1 & 0 & -1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & 0 & -1 \\ 0 & 0 & 0 & \dots & -1 & 0 \end{bmatrix} =$$

$$= \text{triadiag}(0.5,\ 0,\ 0.5)$$

(5) $\rho(B_J) = \max\{|\lambda| : \lambda \in \sigma(B_J)\}$.

(6)

$$\det(\lambda I - B_J) = \begin{vmatrix} \lambda & -0.5 & 0 & \dots & 0 & 0 \\ -0.5 & \lambda & -0.5 & \dots & 0 & 0 \\ 0 & -0.5 & \lambda & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -0.5 & \lambda \end{vmatrix}_{100} \quad \text{row 1 exp} \leftarrow$$

$$= \lambda \cdot (-1)^{1+1} \det(\lambda I_{99} - B_{J\,99}) +$$

$$+ (-0.5) \cdot (-1)^{1+2} \cdot \begin{vmatrix} -0.5 & -0.5 & 0 & \dots & 0 \\ 0 & \lambda & -0.5 & \dots & 0 \\ 0 & 0 & \lambda & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \lambda \end{vmatrix}_{99} \quad \text{col 1 exp} \downarrow$$

$$= \lambda \cdot \det(\lambda I_{99} - B_{J\,99}) + 0.5 \cdot (-0.5)(-1)^{1+1} \cdot \begin{vmatrix} \lambda & -0.5 & \dots \\ & \ddots & \\ & & \lambda \end{vmatrix}_{98}$$

$$= \lambda \cdot \det(\lambda I_{99} - B_{J\,99}) - \frac{1}{4} \det(\lambda I_{98} - B_{J\,98}) =$$

$$= \det(\lambda I_{100} - B_{J\,100}). \quad \Rightarrow$$

(7) $$\boxed{\det(\lambda I_{100} - B_{J\,100}) = \lambda \cdot \det(\lambda I_{99} - B_{J\,99}) - \frac{1}{4} \det(\lambda I_{98} - B_{J\,98})}$$

(7) i) a recurrent formula for

(8) $\sigma(B_{J100}) = \{ \det(B_{J100} - \lambda I_{100}) = 0 \}$.

I think I'm not supposed to do this by software, I mean find all 100 eigenvalues on software and find the max in absolute value for $\rho(B_J)$, I tried 2, 3, 4 by hand though.. Yes, this B_J is having $\rho(B_J) < 1$ always. So yes, Jacobi iterations would converge!

Problem 6.

Show that Gauss-Siedel converges twice as fast as Jacobi iterations for a system of linear equations of order 2 under what conditions.

given

(1) $$Ax = b.$$

Jacobi iteration:

$$x^{(k+1)} = \underbrace{-D^{-1}(L+U)}\, x^{(k)} + D^{-1} b$$

(2) $$x^{(k+1)} = B_J\, x^{(k)} + D^{-1} b = g(x^{(k)})$$

Gauss-Siedel iteration:

$$x^{(k+1)} = \underbrace{-(D+L)^{-1} \cdot U}\, x^{(k)} + (D+L)^{-1} \cdot b$$

(3) $$x^{(k+1)} = B_{GS} \cdot x^{(k)} + (D+L)^{-1} \cdot b = g(x^{(k)})$$

Basically, if we look at RHSs of both (2) & (3) are fixed point iterations in higher dimension. To compare the rate of convergence, here in more dimensional space, we could compare the spectral radii of both B_J & B_{GS} given a

matrix A

(4) $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

By class notes, for this given 2×2 matrix A with (i) only positive entries, we have spectral radii calculated already:

(5) $\rho(B_J) = \sqrt{\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}}}$, for (2) to be convergent $\rho(B_J) < 1 \Rightarrow$

(6) $\rho(B_J) = \sqrt{\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}}} < 1 \overset{a_{ij} > 0}{\Rightarrow}$

(7) $\boxed{a_{12} \cdot a_{21} < a_{11} \cdot a_{22}}$ is the convergent condition for (2).

similarly,

(8) $\rho(B_{GS}) = \frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}} \overset{a_{ij} > 0}{\Longrightarrow}$, for (3) to be convergent $\rho(B_{GS}) < 1 \Rightarrow$

(9) $\rho(B_{GS}) = \frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}} < 1 \Rightarrow a_{ij} > 0$

(10) $\boxed{a_{12} \cdot a_{21} < a_{11} \cdot a_{22}}$ is the convergent condition for (3).

We need to provide conditions when actually

(11) $\boxed{2 \cdot \rho(B_{gs}) = \rho(B_T)}$ provided

(i), (7) and (10)

(12) $2 \cdot \overbrace{\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}}}^{\rho(B_{gs})} = \overbrace{\sqrt{\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}}}}^{\rho(B_T)}$, say $t = \frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}} \Rightarrow$ $(t > 0)$ as $a_{ij} > 0$

(13) $2 \cdot t = \sqrt{t} \iff 4t^2 = t \iff$

$t(4t-1) = 0 \iff \boxed{t = \frac{1}{4}}$

So

(14) $\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}} = \frac{1}{4}$.

To conclude, it is a must for A to have

- (i): $\boxed{a_{ij} > 0}$ $\forall\ i, j = 1, 2$.
- (7) & (10): $\boxed{a_{12} \cdot a_{21} < a_{11} \cdot a_{22}}$, and finally
- (14): $\boxed{\frac{a_{12} \cdot a_{21}}{a_{11} \cdot a_{22}} = \frac{1}{4}}$ $\Rightarrow$ with this

we can skip (7) & (10)!