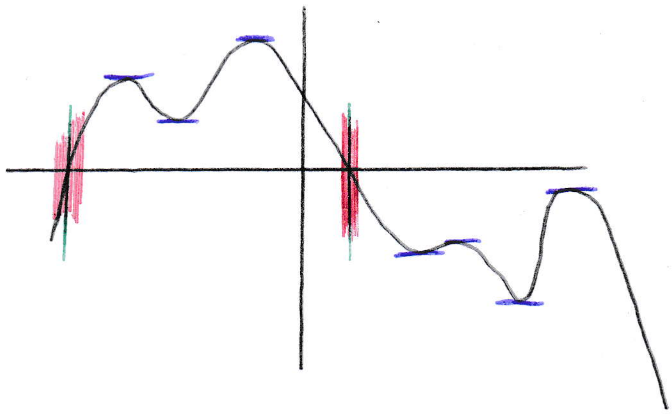
01/11/25

LRCS - VI

07/11/25

Locating Roots of a polynomial of any degree VIA change in sign between the function's stationary points



Gavin Simpson - @gdsimpson3

Locating roots VIA change in sign

(1) input : 7x^5 + 9x^4 - 3x^2 + 4x + 2

input rules
→ Ascending powers
→ missing powers allowed
→ Ax^(Exponent)

split through ("+" || "-")

→ ["7x^3" ......] → [[A, Exponent]..

7x^3 → 7, ^3 → [7, 3]

if no x → [2, 0]

if no ^ → [7, 1]

→ [[7, 5], [9, 4], [-3, 2], [4, 1], [2, 0]]

missing powers → won't be needed

(2) <u>Creating f(x)</u>

f(x) = (7 * (x ** 5)) + (9 * (x ** 4))....

Loop & sum to variable

(3) <u>find min & maxes to check if there are roots</u> 1/2

why?: if func has no roots, program will crash, find min & max & see if it changes sign indicating roots

(3.1) $\frac{d^n y}{dx^n}$ → [7, 5] → [7 * (5), 5-1]

not needed ← ∫ → [7, 5] → [7 / (5+1), 5+1]

loop & create according array
→ if 0, remove

(3.2) technically only need stationary points, nature doesn't really matter?

scenario 1 → 1 stationary point, nature : min → condition for root it must be ⊖

scenario 2 → 8 stationary points → to know root or not, find y of any two & see if crosses via sign change

3
find min/max to check if there are roots 2/2

Scenario 3 → No stationary points → what means: straight line

3.2.1
Programatically analyse if function has roots VIA stationary point count:

cases : 0 , 1 , n

0 → straight line
1 → check position & two trailing edges
n → check all roots & pass if diff sign's found

3.3 finding stationary points

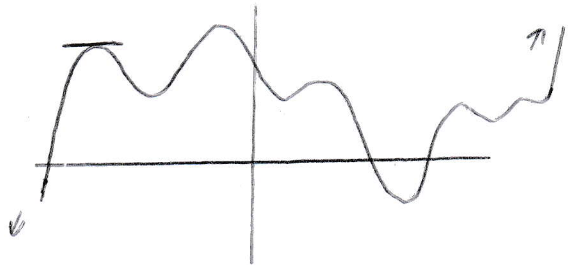
roots of $f'(x)$, recursion ? !potential!

3.1.2 Derivative logic

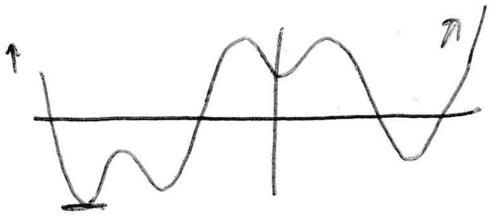
[7,5] → [7*5 , ~~5+1~~ 5-1]

[2,0] → [2*0 , 0-1] → musn't let this pass

4 finding Roots



1 → find earliest stationary point and keep going backwards till theres a change in sign
2 → then switch direction & go in smaller till next change
3 → carry on till 0 or certain n.o. of d.p.

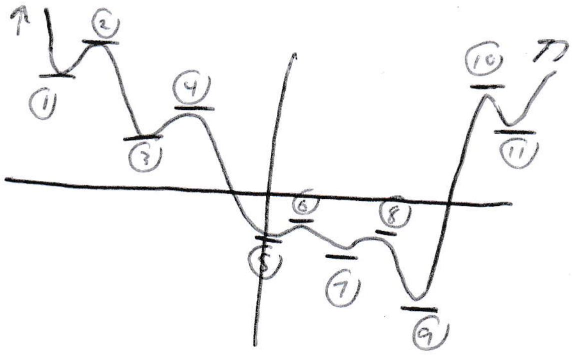


4.1
for n stationary points → option 1 → go left to right & look for sign changes

start with first stationary point and go back, conduct Rootloc, then go forwards till sign change and Root loc

4.2 finding 2 stationary points with root between

- Loop through all points & sign change pair



- 1 & 2 → no sign change
- 1 & 3 → no
- 1 & 4 → no
- 1 & 5 → yes

- 1 & 2 → no
- 2 & 3 → no
- 3 & 4 → no
- 4 & 5 → yes

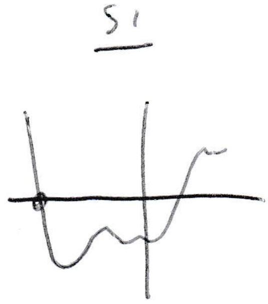
4.3 first root

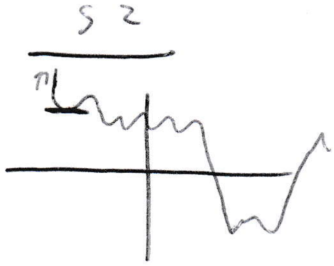
- get nature of point.

if min & above x axis, then search ~~les~~ Right

if max & below x axis, look Right

similarly go left else

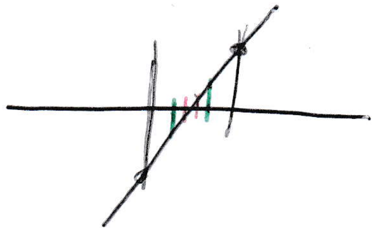




4 Root locating

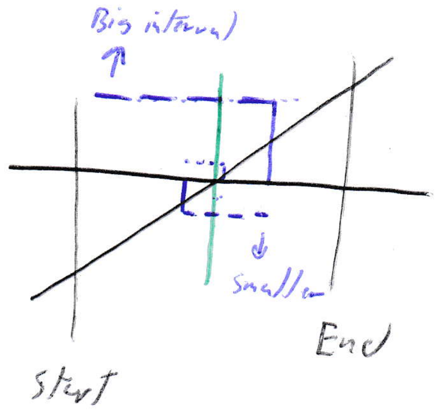
go from A to B in big intervals, once sign changes, go back in smaller intervals

Repeat till u find Root to certain D.P



④ Root locating - Programatically

```
'''
true = positive
False = Negative '''
```



(*)' Interval = 2 ** (- (iteration count))

Direction = ~~1~~ (-1) ** (iteration count)

↳ swap directions every iteration
used in for loop Range steps
(direction * interval)

Concept: start → $\underbrace{\text{sign change}}_{\text{new start}}$
↓
$\underbrace{\text{Sign change}}_{\text{start}}$ ← other direction
→ change....

↓ interval smallens & direction swaps
+1 iteration count

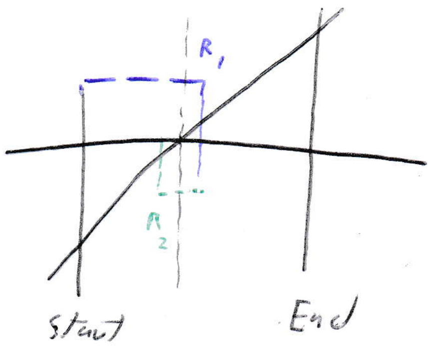
Psudeocode

```
Start = ....
End = .....
Start Sign = true
End Sign = False
```

for Root in range(s, e, interval)
going from S to E in interval

sign change either start works

```
if (Root Sign != Start sign) or (Root Sign == End sign)
    ~~# Root is new start~~
    ~~start = Root~~
    ~~Direct~~ iteration count += 1   → changes interval & direction (*)'
# Next loop must go back from Root to start
# loop after must go from Next to Root
```



Start → end
~~← R~~
R_1 → start
R_2 → R_1

swap start & end
R_n is new start

Locating Roots VIA change in sign

④ Root locating - Psudeocode

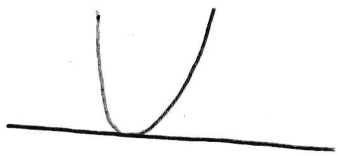
true = Positive
False = Negative

```
Start = ....
End = ....
start Sign = ....
End Sign = ...
interval = 2 ** (-1 * IterationCount)
direction = (-1) ** (iterationCount)
iterationCount = 0
D.P Accuracy = ...     # when to stop

while D.P Accuracy not met:          <- or
    [for Root in Range (start, end, interval * direction):
        If (Root Sign != Start Sign):
            End = start
            start = Root
            incremented = +1
```

Steps when $f(x)=0$

Limitation: this mechanism would work in most cases except Roots that touch



workaround: A touching Root is a stationary point... Hence all stationary points could just be checked

Python limitation: Range(), increment steps cannot be Type int

Custom loop →

```
Root = start
while Root < end:
    Root += interval
```

doesn't show condition

```
Condition = .....
if direction > 0:
    condition = ....
else
    ....
```

change condition

c1 = start < End
c2 = start > end

Locating Roots VIA change in sign

4 Custom loop

flow : 1 Start(S) →(4) end(E) : Pass Root @ R1

2 start(E) ←(−) R1(S) = R1 → start

3 R2(S) → R1(E)

conditions : 1 Root = start

while Root < End .... Root ++

2 Root = R1

~~start = End~~ End = start

while Root > End .... Root --

3 Root = R2

End = R1

while Root < End .... Root ++

var Reassigns:

1 Root = start
End = End

2 Root = R1
End = start
direction = Back

3 Root = R2
End = R1

we can have toggle for directions, true = →, false = ←

~~while True~~

start = start Param

End = End Param

interval = 2 ** (-1 * iteration count)

direction = (-1) ** (iteration count)

iteration count = 0

*

1 start = start
Root = start
End = End
direction = →

2 Root = R1
End = start
start = Root
direction = ←

3 Root = R2
End = start (R1)
start = Root

Locating Roots VIA change in sign

Overall flow

$f(x)$

↓

$\frac{dy}{dx}$

↓

Stationary points

↓

Check if any point is a root

↓

Look for sign changes between Roots & create pairs

↓

find Root ~~convergence~~ between pairs VIA convergence

↓

then look at Edge Roots

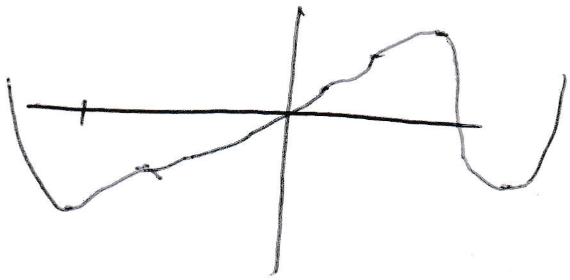
Locating Roots VIA change in sign

finding stationary point pairs

~~[9, 8, 4,~~

[9, 7, 5, 3, -2, -4]

f(9) = −, f(7) = +, f(5) = +, f(3) = +, f(-2) = −, f(-4) = −



Best to take from left

[-4, -2, 3, 5, 7, 9]

− − + + + −

for point in Range ( )

if not fn(point) sign == fn(Next point) sign:

Root between them

make sure to exclude stationary Roots from pair finding

Locating Roots VIA change in sign

finding stationary points

Scenario 1: 4 deg polynomial

$f(x) = x^4 \ldots\ldots$

$f'(x) = x^3$

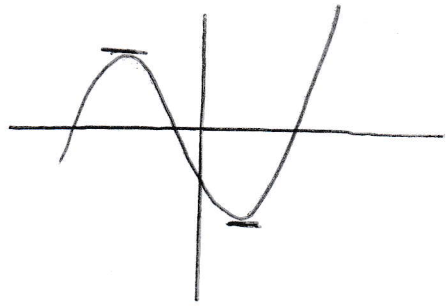
Root finder $(f'(x))$

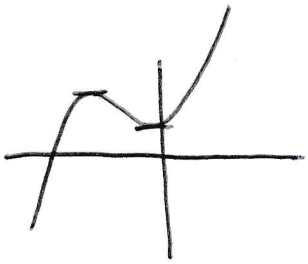
↳ $f''(x)$

Root finder $(f''(x))$

↳ ...

cubic





Theoretically if Root finder returns true Roots, we don't need to worry about its long recursion

if $f'(x)$ is 2nd degree, we can use the quadratic formula for the Roots which will give ~~use~~ us the 2 stationary points of a cubic. This can be the end of Recursion

Input: 3rd degree

Stat point will Jump @ deg 2 & channel it to Quadratic

Locating Roots VIA change in sign

Root finder function

param: $f(x)$

derivative $\Rightarrow f'(x)$

stationary point finder $\Rightarrow [[\alpha, \beta], [\gamma, \delta]]$

$\downarrow$

Pairs

& edge Roots

signchange func $(\alpha, \beta) \rightarrow$ Roots[]

edge Root (α, ±inf) $\rightarrow$ Roots[]

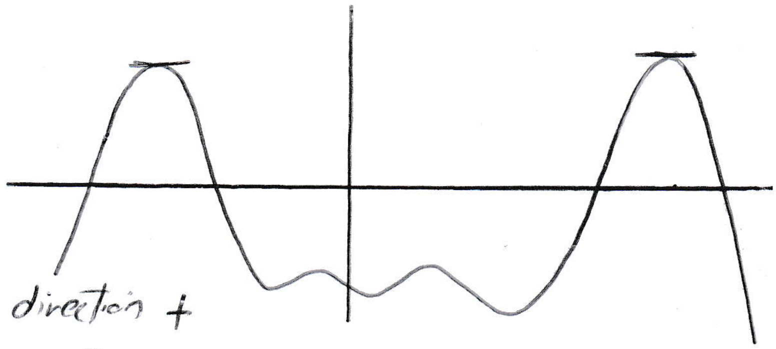
Return Roots

Locating Roots VIA change in sign

Edge Roots

Stationary Point [0]

Stationary Point [-1]



Pairs:

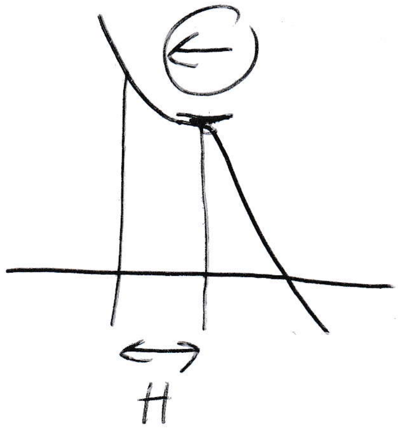
direction - ↑   direction + ↑

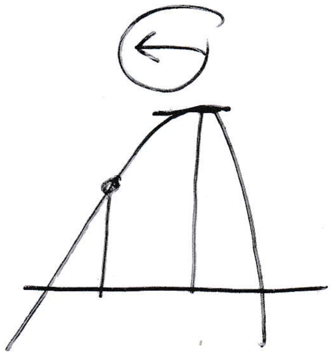
[[SP[0], -inf], [SP[-1], +inf]]

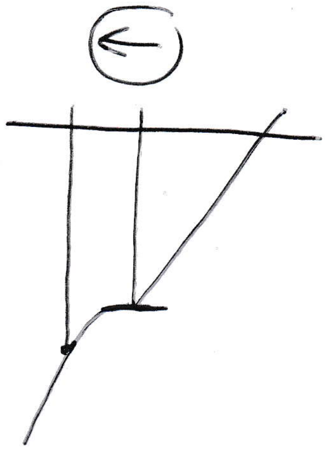
Checks: if ~~SP[0]~~ max f > 0 → yes Root

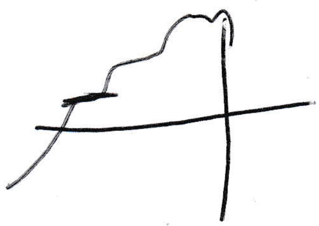
min f < 0 → yes Root

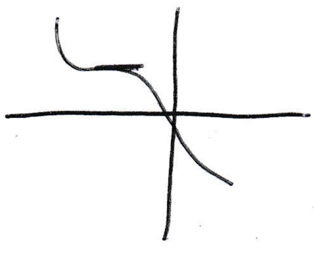
dealing with inflection → look either side







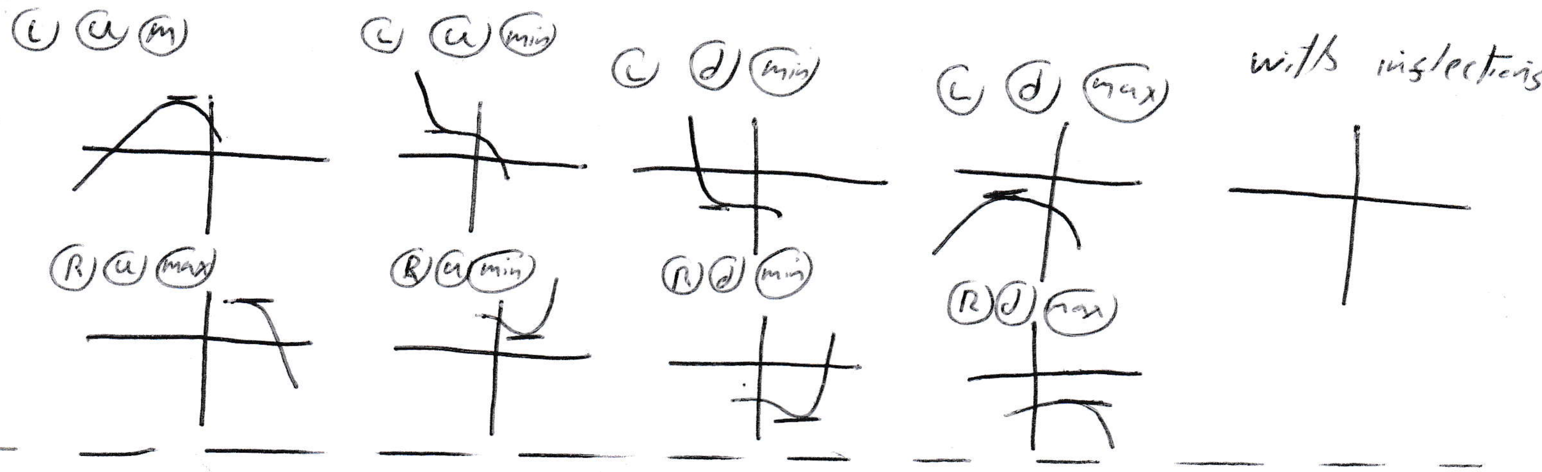




Locating Roots VIA change in sign

Edge Roots 2

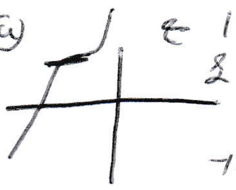
Scenarios



conditions → (L): u & m → look left
(L): u & min → ignore
(L): D & max → ignore
(L): D & min → look left
(R): u & max → look Right
(R): u & min → ignore
(R): d & max → ignore
(R): d & min → look Right

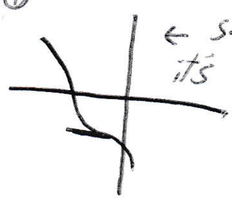
with inflection points → we wont know if its a max or min

(L)(u)



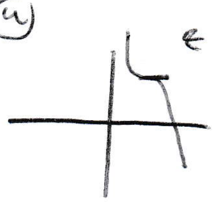
← look from point & look left, if its going down then theres Root

(L)(d)

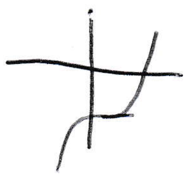


← Same, since its (d), its its closer to x axis

(R)(u)



←



Classification of nature could take inflection & call max/min dependent on direction its ~~go~~ from - add 100 in direction & see if going up or down