5. c)

Avand in vedere ca algoritmul este descris din prisma unei relatii de recurenta, invariantul este chiar acea relatie de recurenta.

A picture containing text, whiteboard

Description automatically generated

a , k = 0

I = , k > 0

Voi implementa acest algoritm in mod iterativ:

Function eps(a, e):

actual <- a

next <- ((2\*actual) + a/(actual\*\*2))/3

if abs(next - actual) < e:

Scrie next

<iesi din functie>

else:

while abs(next - actual) >= e:

Scrie next

actual <- next

next <- ((2 \* actual) + (a / (actual ^ 2))) / 3

**(i)**Invariantul este corect inainte de intrarea in ciclu, deoarece variabila “actual” stocheaza valoarea primului termen din suma, lucru care trebuie sa se intample cu primul termen.

**(ii)**Invariantul e corect pe parcursul ciclului, deoarece variabila “actual” stocheaza suma termenilor pentru care modulul diferentei dintre termenul initial si cel din spate e mai mica decat eroarea epsilon., iar noi ne dorim acest lucru.

**(iii)**Invariantul este corect la iesirea din ciclu, deorece invariantul implica postconditia, ciclul “while” se opreste la primul termen gasit pentru care modulul diferentei dintre el si termenul anterior este mai mica decat eroarea epsilon, lucru pe care ni-l dorim.

In acest sens, putem urmari starea variabilei actual pe parcursul ciclului:

Diagram, letter

Description automatically generated

**(iv)**Finitudinea

Finitudinea depinde de precizia cu care vrem rezultatul, cand aceasta este mare, vor aparea limitarile calculatorului.

Finitudinea ar putea depinde de valoarea variabilei “actual”, acesta este descrescatore, iar functia care depinde de ea va fi descrescatoare, cu cat termenul va fi mai mic, cu atat diferenta pana la valoarea epsilon va fi mai mica.

**F(p) = current –**  (numarul de zecimale prin care trebuie sa trecem pentru a ajunge strict mai mic ca )