Algoritmi fundamentali

Curs 10

Algoritmi matematici/numerici (si nu numai)

Dr. ing. Kiss Istvan

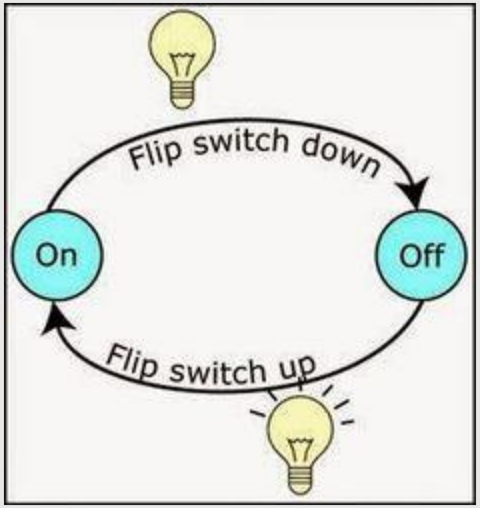
istvan.kiss@umfst.ro

Cuprins

1. Masina de stare finita (finite state machine - FSM)

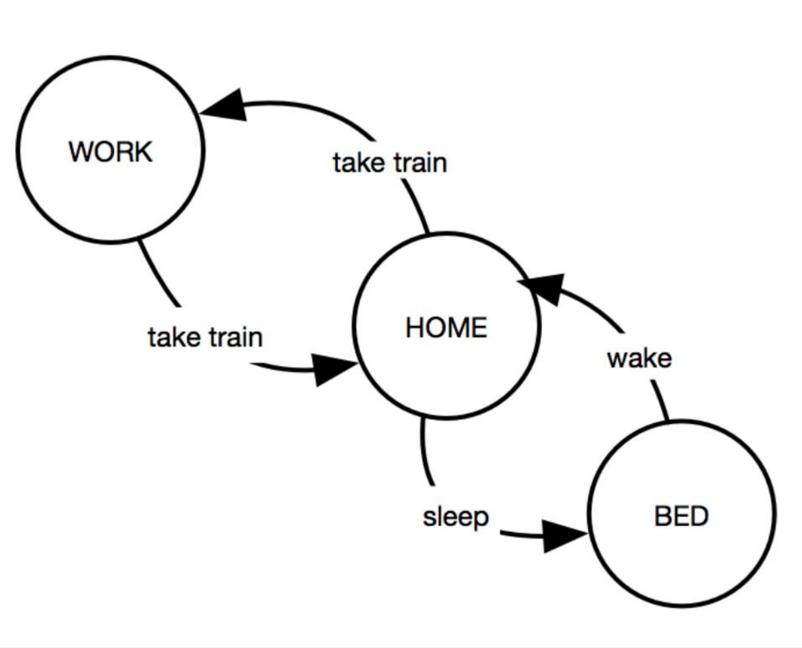
- Exemple:
- Testare par/impar
- Evenimente: activare de stari pe baza tastelor, pe baza datelor receptionate prin retea, etc...
- Deschidere usa...
- Comanda liftului???

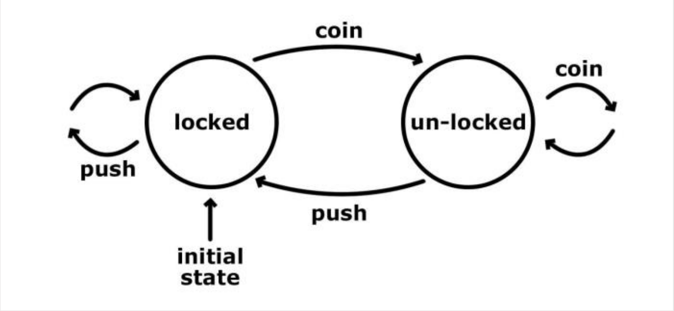
- **Tabel cu tranzitiile dintr-o stare in alta**

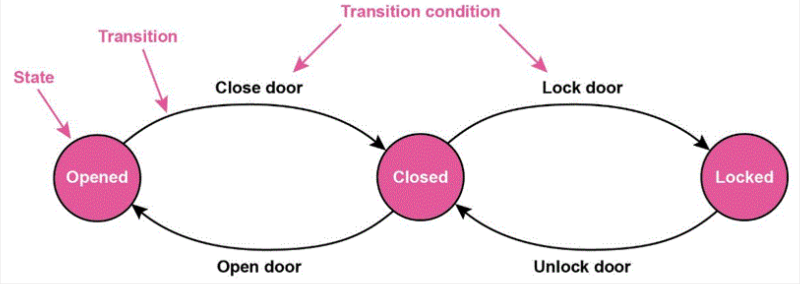


1. Masina de stare finita (finite state machine - FSM)

- **Is a mathematical model of computation.**
- **It is an abstract machine that can be in exactly one of a finite number of *states* at any given time.**







1. Masina de stare finita (finite state machine - FSM)

- **Is a mathematical model of computation.**
- **It is an abstract machine that can be in exactly one of a finite number of *states* at any given time.**

- **Exemplu: tabel de tranzitii**

Current State	Next State (δ)		Output(λ)
	0	1	
q_0	q_1	q_2	1
q_1	q_2	q_1	1
q_2	q_2	q_0	0

2. Algoritmi numerici

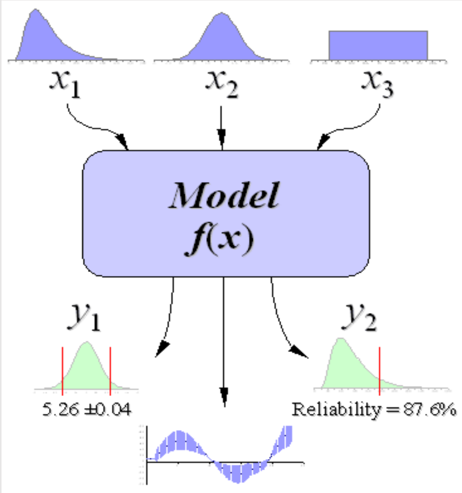
- Algoritmii numerici prelucreaza numere!
- Ex.: generare numere aleatoare, determinare factori primi, CMMDC, calculare arii geometrice, etc.
- Avansate: algoritmi adaptivi, simulare prin metoda Monte Carlo, tabele intermediare...

2. Algoritmi numerici

- **Simulare prin metoda Monte Carlo**
- domeniu larg de aplicabilitate
  - aproximarea numarului pi
  - analiza riscurilor si impactului
  - electronica, dinamica fluidelor, telecomunicatii, procesarea semnalelor/imaginilor...
  - si multe altele...

- **Se bazeaza pe “randomizare” pentru a obtine solutii numerice**

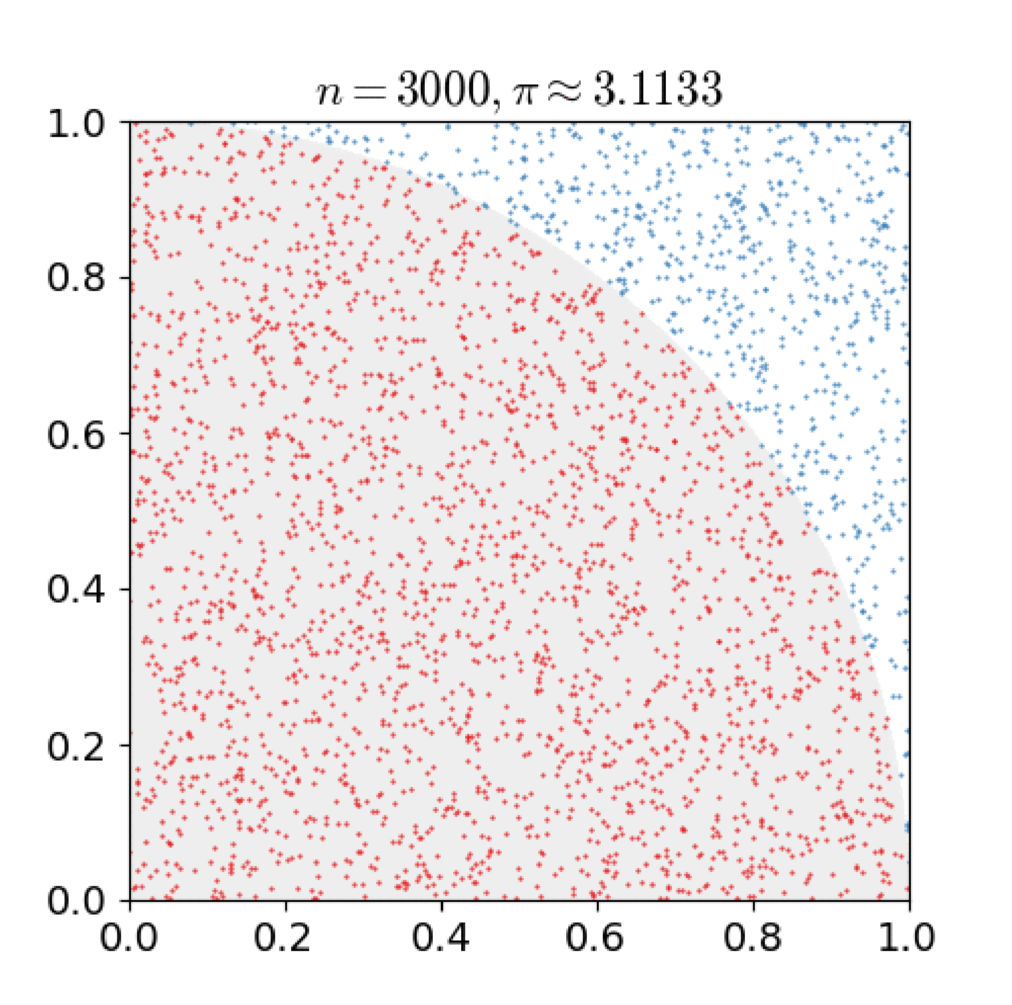
2. Metoda Monte Carlo



- Utilizeaza multimea numerelor generate aleator pentru a rezolva problem in general deterministice.
- Utilizate in general in cazul problemelor de optimizare, integrare numerica.
- In principiu, metoda poate fi utilizata in a rezolva orice problema de natura deterministica.
- Metodele MC au in comun:
  1. se defineste domeniul intrarilor
  2. se genereaza seturi de intrari in mod aleator pe baza unei distributii probabilistice
  3. se efectueaza calcule deterministice pe baza intrarilor
  4. se executa agregarea rezultatelor

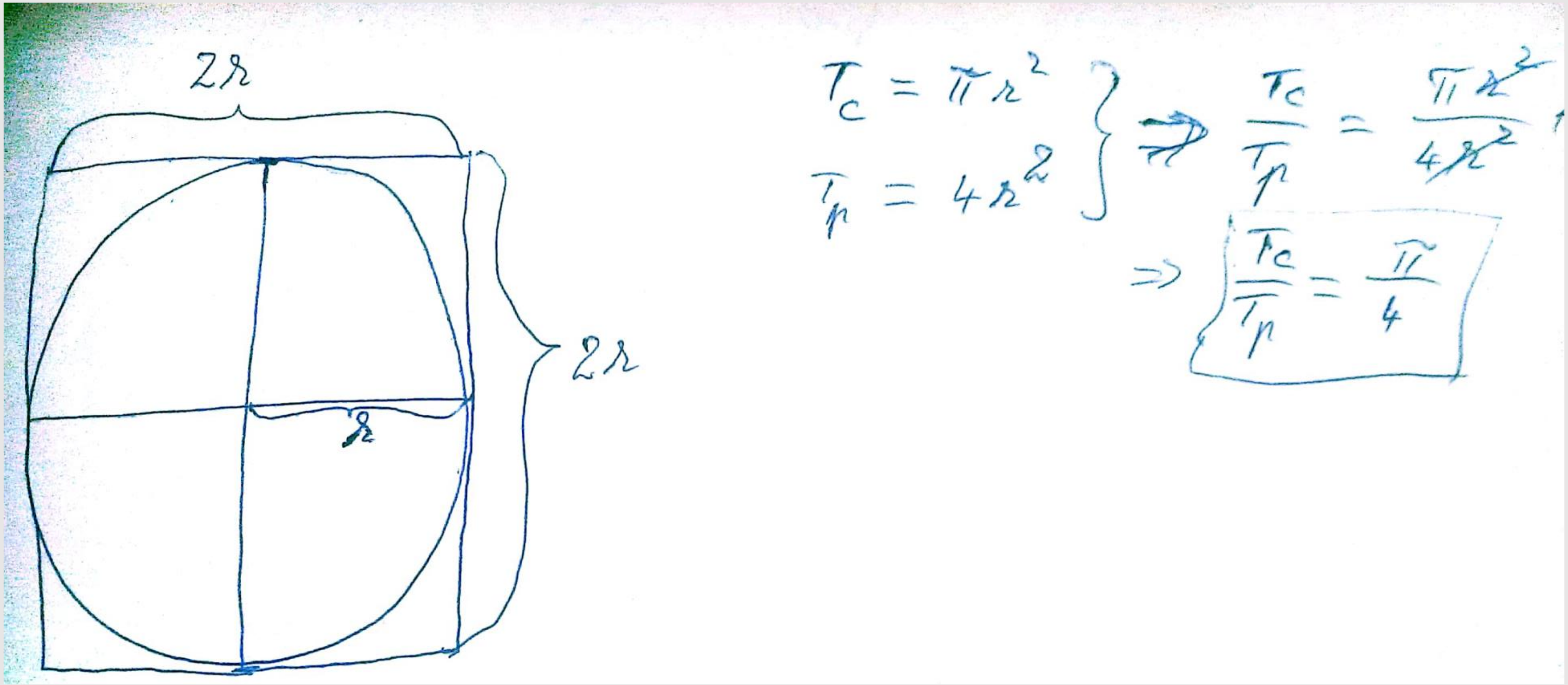
2. Metoda Monte Carlo, ex:

- Stiind ca raport cerc si patrat este pi/4 se poate aproxima pi prin simpla generare de puncte in domeniul bidimensional.
- Cu cat generam mai multe puncte, cu atat aproximarea este mai precisa.



2. Metoda Monte Carlo, ex:

- Stiind ca raport cerc si patrat este pi/4 se poate aproxima pi prin simpla generare de puncte in domeniul bidimensional.
- Cu cat generam mai multe puncte, cu atat aproximeare este mai precise.



2. Generare numere aleatoare (Randomizare)

- Are un rol important in multe aplicatii.
- Permite simulare de procese random, testare automata de algoritmi cu date random.
- De ex.: Integrare numerica Monte Carlo, care selecteaza in mod aleator puncte pentru a estima aria geometrica.

- **Orice algoritm bazat pe numere random necesita un generator random!!!**

2. Generare numere aleatoare (Randomizare)

- Nu exista algoritm de calculator care sa genereze numere pur random!
  - **daca algoritmul este cunoscut si se stie starea actuala atunci exista sansa sa putem *prezice* urmatorul numarul aleator.**

- **In aplicatii sensibile de numere random (ex. Loto, criptare…), se folosesc surse externe (masuratori radiatii, semnale radio,...) – www.random.org**

2. Generare numere aleatoare (Randomizare) - PRNG

- **1. Metoda comuna: linear congruential generator**

$$X_{n+1} = (A \times X_n + B) \text{ Mod } M$$

- A, B, M sunt constante.
- X_0 initializeaza generatorul; secventa depinde de acest numar (denumit en.: seed)
- dupa M numere secventa se va repeta...
- ex.: A=7, B=5, M=11, X_0=0;
  - 0,5,7,10,9,2,8,6,3,4...
- Numerele generate sunt foarte predictibile!!!
  - alte metode mai complexe care foloses mai multe PRNG...

3. Descompunere in factori primi

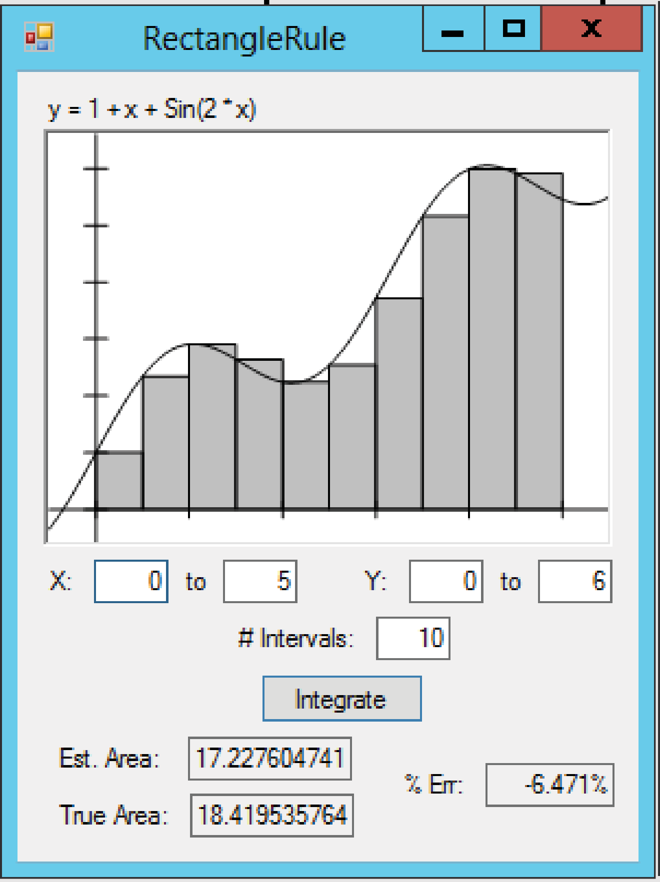
```
List Of Integer: FindFactors(Integer: number)
        List Of Integer: factors
        Integer: i = 2
        While (i < number)
                // Pull out factors of i.
                While (number Mod i == 0)
                        // i is a factor. Add it to the list.
                        factors.Add(i)
                        // Divide the number by i.
                        number = number / i
                End While
                // Check the next possible factor.
                i = i + 1
        End While
        // If there's anything left of the number, it is a factor,
too.
        If (number > 1) Then factors.Add(number)
        Return factors
End FindFactors
```

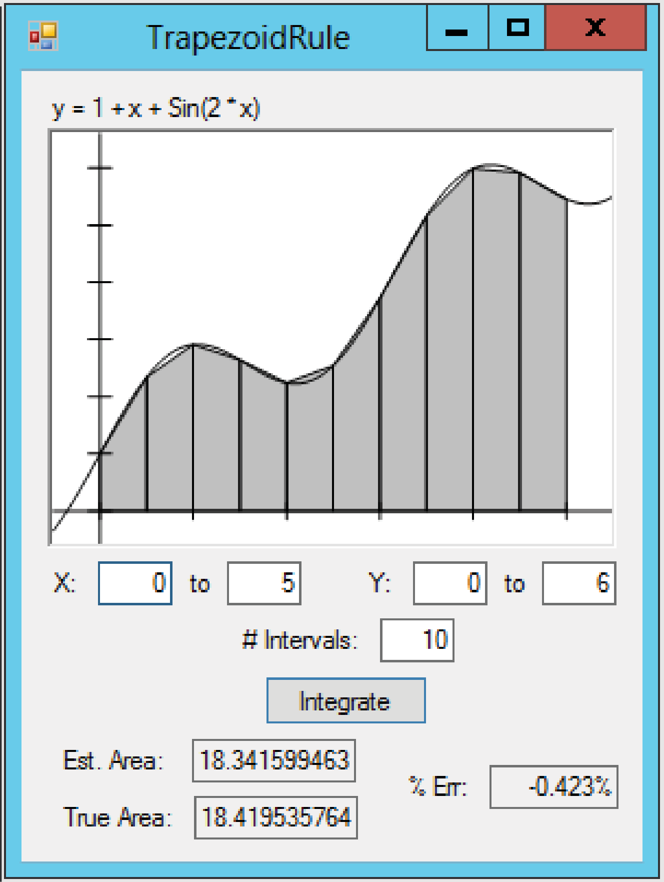
4. Integrare numerica

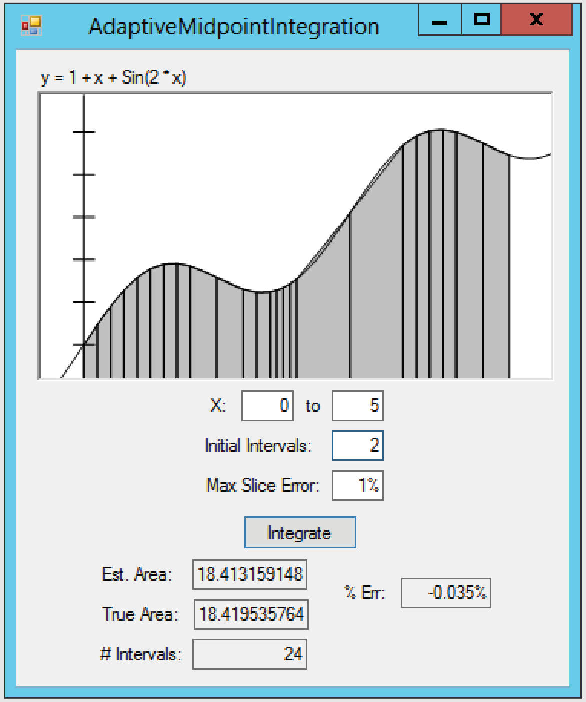
- Aproximarea ariei sub o curba definite de o functie.
- In general, y=f(x), deci rezultatul este o arie bidimensionala.
- Se aplica in cazul functiilor complexe pentru care calculul anti-derivatei este greu posibila
- Sau in cazul cand in locul functiei avem un process fizic pentru care nu se poate determina un model (functie) matematic.

4. Integrare numerica

- Metode: prin formule Newton-Cotes (polinoame)
  - formula dreptunghiurilor
  - formula trapezelor
  - quadratic adaptiv







4. Integrare numerica

```
Float: UseRectangleRule(Float: function(), Float: xmin,
Float: xmax,
      Integer: num_intervals)
      // Calculate the width of a rectangle.
      Float: dx = (xmax - xmin) / num_intervals
      // Add up the rectangles' areas.
      Float: total_area = 0
      Float: x = xmin
      For i = 1 To num_intervals
            total_area = total_area + dx * function(x)
            x = x + dx
      Next i
      Return total_area
End UseRectangleRule
```

4. Integrare numerica

```
Float: UseTrapezoidRule(Float: function(), Float: xmin, Float: xmax,
        Integer: num_intervals)
        // Calculate the width of a trapezoid.
        Float: dx = (xmax - xmin) / num_intervals
        // Add up the trapezoids' areas.
        Float: total_area = 0
        Float: x = xmin
        For i = 1 To num_intervals
                total_area = total_area + dx * (function(x) + function(x + dx)) / 2
                x = x + dx
        Next i
        Return total_area
End UseTrapezoidRule
```

4. Integrare numerica

```
// Integrate by using an adaptive midpoint trapezoid rule.
Float: IntegrateAdaptiveMidpoint(Float: function(),
Float: xmin, Float: xmax, Integer: num_intervals,
Float: max_slice_error)
         // Calculate the width of the initial trapezoids.
         Float: dx = (xmax - xmin) / num_intervals
         double total = 0
         // Add up the trapezoids' areas.
         Float: total_area = 0
         Float: x = xmin
         For i = 1 To num_intervals
                  // Add this slice's area.
                  total_area = total_area +
                           SliceArea(function, x, x + dx, max_slice_error)
                  x = x + dx
         Next i
         Return total_area
End IntegrateAdaptiveMidpoint
// Return the area for this slice.
Float: SliceArea(Float: function(),Float: x1, Float: x2,
Float: max_slice_error)
```

4. Integrare numerica

```
    // Calculate the function at the endpoints and the midpoint.
    Float: y1 = function(x1)
    Float: y2 = function(x2)
    Float: xm = (x1 + x2) / 2
    Float: ym = function(xm)
    // Calculate the area for the large slice and two subslices.
    Float: area12 = (x2 - x1) * (y1 + y2) / 2.0
    Float: area1m = (xm - x1) * (y1 + ym) / 2.0
    Float: aream2 = (x2 - xm) * (ym + y2) / 2.0
    Float: area1m2 = area1m + aream2
    // See how close we are.
    Float: error = (area1m2 - area12) / area12
    // See if this is small enough.
    If (Abs(error) < max_slice_error) Then Return area1m2
        // The error is too big. Divide the slice and try again.
    Return
        SliceArea(function, x1, xm, max_slice_error) +
        SliceArea(function, xm, x2, max_slice_error)
End SliceArea
```

5. Operatii cu polinoame

- adunare, inmultire

```
for(i=0;i<2*n-1;i++) r[i]=0;
for(i=0;i<n;i++)
        for(j=0;j<n;j++)
                r[i+j]+=p[i]*q[j];
```

6. Operatii cu matrici

- adunare, inmultire

$$\begin{pmatrix} 1 & 3 & -4 \\ 1 & 1 & -2 \\ -1 & -2 & 5 \end{pmatrix} \begin{pmatrix} 8 & 3 & 0 \\ 3 & 10 & 2 \\ 0 & 2 & 6 \end{pmatrix} = \begin{pmatrix} 17 & 25 & -18 \\ 11 & 9 & -10 \\ -14 & -13 & 26 \end{pmatrix}$$

```
for (i = 0; i < N; i++)
  for (j = 0; j < N; j++)
    for (k = 0, r[i][j] = 0; k < N; k++)
      r[i][j] += p[i][k]*q[k][j];
```

5. Probleme - optional

1. Testarea metodelor studiate cu numere concrete si prin implementare intr-un limbaj de programare.