Ernest SCHEIBER

JAVA ÎN CALCULUL ŞTIINŢIFIC

Braşov

2016

Cuprins

Prefaţă

Lucrarea de faţă are ca obiectiv problematica legată de implementarea în Java a aplicaţiilor de calcul ştiinţific şi este o versiune actualizată a lucrării noastre, [10].

Din partea cititorului vom presupune existenţa unor abilităţi de programare în Java şi bineînţeles, cunoştinţe elementare de calcul numeric. Anumite părţi din lucrare cer cititorului familiarizarea cu un anumit produs informatic. Pentru fiecare produs utilizat, Internetul oferă o mulţime de informaţii privind instalarea, utilizarea, exemple, interfaţa de programare, răspunsuri la întrebări frecvente, etc.

Implementarea unei metode de calcul numeric bazat pe un algoritm iterativ este exemplificată prin crearea unei mini-biblioteci de programe numerice care este folosit pe parcursul lucrări. Pentru configurarea şi executarea sarcinilor se va utiliza cu precădere *apache-ant.*

Vom pune în evidenţă posibilitatea utilizării în Java a resurselor oferite de produsele matematice *Mathematica, Maple* şi *Scilab.*

Utilizarea limbajului de programare Java în calculul ştiinţific este reprezentată prin biblioteci de clase pentru rezolvarea unor probleme specifice domeniului. Lucrarea exemplifică doar folosirea pachetele *apache commons-math* şi *Jama.*

O evidenţă şi observaţii comparative a resurselor de calcul ştiinţific se găsesc pe Internet [13, 14].

Utilizarea unei funcţii definită printr-un şir de caractere este o problemă de programare specifică unei aplicaţii de calcul numeric care se doreşte utilizată prin intermediul unei interfeţe grafice sau în mediu distribuit.

Interfeţele grafice şi aspecte privind vizualizarea unor obiecte matematice sunt de asemenea atinse. Este dat un exemplu de interfaţă grafică programată în JavaFX, pachet dezvoltat în ultimii ani, parte din distribuţia Java Development Kit - JDK. Se prezintă soluţii pentru vizualizarea graficului unei funcţii, prin intermediul unor produse specializate, integrabile în aplicaţii Java.

Tipurile de aplicaţii distribuite tratate în lucrare sunt aplicaţiile Web bazate pe servlet şi serviciile Web dezvoltate pe modelul apelului de procedură la distanţă (Remote Procedure Call) dar şi serviciile REST. Exemplificările utilizează pachetele *metro* şi respectiv, *jersey.* Prin utilizarea pachetului *apache commons-fileupload* este dată o soluţie pentru încărcarea unui fişier.

În prezent calculatoarele uzuale permit efectuarea de calcule în paralel prin unitatea de procesare grafică (GPU). Exemplificăm tehica de programare în Java prin produsul *Aparapi,* bazat pe *OpenCL.*

Codurile tuturor programelor din lucrare pot fi descărcate de la adresa

https://github.com/e-scheiber/Scientific_Computing_and_Java.git.

Partea I

Programarea aplicaţiilor numerice în Java

Capitolul 1

Aplicaţii numerice simple

Scopul acestui capitol este prezentarea unui mod de implementare în limbajul de programare Java a unor metode numerice. Se vor utiliza mai multe produse informatice ajutătoare dar nu vom face apel la un mediu integrat de dezvoltare (*Integrated Development Environment* - IDE). Motivaţia acestei opţiuni este simplă: un mediu integrat de dezvoltare acoperă (ascunde, face transparentă) multe din activităţile care se întreprind. În plus, din punct de vedere educativ, nu dorim să promovăm un anumit mediu integrat de dezvoltare în dauna altuia.

Pentru început ne propunem să calculăm:

1. Soluţia negativă a ecuaţiei $2^x - x^2 = 0$.

   Ecuaţia are 3 soluţii $x_1 \approx -0.7666647 \in (-1, -\frac{1}{2}), x_2 = 2, x_3 = 4$;

2. Integrala $\int_0^{\frac{\pi}{4}} \ln(1 + \tan x)\mathrm{d}x = \frac{\pi}{8} \ln 2 \approx 0.2721983$;

probleme pentru care se vor realiza programe simple. Apoi aceste programe vor fi dezvoltate pentru a putea trata un caz general şi pentru a putea fi utilizate în mediu distribuit.

Rezolvarea numerică a unei probleme de calcul numeric conduce de multe ori la construirea unui şir $(x_k)_{k \in \mathbb{N}}$, despre care se arată că converge într-un sens către soluţia problemei. Rezultate privind evaluarea erorii, a vitezei de convergenţă, a complexităţii unui algoritm sunt factori care prezintă importanţă în alegerea metodei de rezolvare numerică a unei probleme.

Aspectele care ne interesează privesc implementarea metodelor de rezolvare numerică utilizând limbajul de programare Java.

1.1 Pseudocodul unui algoritm iterativ

Una dintre proprietăţile unui algoritm este finitudinea. Pentru aceasta este nevoie de o regulă de oprire. Spre exemplificare, în cadrul unei probleme numerice, o regulă de oprire simplă este:

Dacă eroarea relativă (absolută) a aproximaţiei $x^{k+1} = Y$ pentru $x^k = X$ este mai mică decât un număr pozitiv *eps*, sau dacă numărul de iteraţii executate *ni* este egal cu

numărul maxim admis de iteraţii *nmi* atunci programul se opreşte; altfel se trece la o nouă iteraţie.

După oprirea calculelor, se poziţionează un indicator de răspuns *ind* pe 0, dacă eroarea relativă este mai mică decât *eps*, iar în caz contrar pe 1.

Numărul *eps* este denumit toleranţă de calcul sau test de precizie.

Natura problemei poate impune şi alte condiţii pentru terminarea procesului de calcul iar valoarea parametrului *ind* va preciza felul în care a avut loc oprirea calculului.

Pseudocodul algoritmului metodei iterative este

Algorithm 1 Pseudocodul metodei iterative

```
1: procedure METODA ITERATIVĂ
2:     generarea aproximaţiei iniţiale Y
3:     ni ← 0
4:     do
5:         ni ← ni + 1
6:         X ← Y
7:         generarea aproximaţiei următoare Y
8:         d ← ||Y−X|| / ||X||
9:     while (d ≥ eps) şi (ni < nmi)
10:    if d < eps then
11:        ind ← 0
12:    else
13:        ind ← 1
14:    end if
15:    return Y
16: end procedure
```

În Java există posibilitatea ca în locul limitării numărului maxim admis de iteraţii să fie fixată durata maximă admisă de timp de calcul [9]. Această abordare utilizează programare concurentă prin fire de execuţie, tematică ce nu face obiectul acestei lucrări.

1.2 Metode numerice pentru rezolvarea problemelor

Calculul soluţiei negative a ecuaţiei $2^x - x^2 = 0$

Determinarea soluţie negative a ecuaţei $f(x) := 2^x - x^2 = 0$ se va face utilizând metoda tangentei [5, 3]:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{1.1}$$

Dacă aproximaţia iniţială x_0 este aleasă într-o vecinătate *convenabilă* a unei soluţii, atunci şirul (1.1) converge către aceea soluţie.

Aproximaţia iniţială o vom alege $x_0 = -0.5$.

Derivata $f'(x)$ a funcţiei date de membrul stâng al ecuaţiei, într-un punct, va fi calculată numeric utilizând extrapolarea Richardson, [3]. Se construieşte tabloul subdiagonal

$$\begin{array}{cccc} D_{0,0} & & & \\ D_{1,0} & D_{1,1} & & \\ \vdots & \vdots & \ddots & \\ D_{m,0} & D_{m,1} & \ldots & D_{m,m} \end{array}$$

folosind formulele de recurenţă

$$\begin{aligned} D_{k,0} &= \frac{f(x+\frac{h}{2^k}) - f(x-\frac{h}{2^k})}{\frac{h}{2^k}} \qquad k = 0, 1 \ldots, m; \\ D_{k,j} &= \frac{4^j}{4^j-1} D_{k,j-1} - \frac{1}{4^j-1} D_{k-1,j-1} \qquad \begin{array}{l} j = 1, 2, \ldots, m; \\ k = j, j+1, \ldots, m. \end{array} \end{aligned}$$

Aproximaţia derivatei va fi $D_{m,m}$.

Datele de intrare pe care utilizatorul - clientul - trebuie să le furnizeze sunt:

- expresia lui $f(x)$;
- aproximaţia iniţială;
- toleranţa;
- numărul maxim admis de iteraţii.

Datele de ieşire pe care le aşteptăm să le primim de la programul de calcul sunt:

- indicatorul de răspuns al programului;
- aproximaţia soluţiei obţinute;
- valoarea membrului stâng al ecuaţiei calculată în soluţia aproximativă obţinută;
- numărul iteraţiilor efectuate.

De multe ori prezintă interes urmărirea evoluţiei rezultatelor intermediare. Accesul la aceste date va fi o cerinţă a programelor ce vor fi dezvoltate.

Calculul integralei $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x$

Pentru calculul integralei se va utiliza o schemă adaptivă bazată pe metoda lui Simpson [11]

$$\int_a^b f(x)\mathrm{d}x = \frac{b-a}{6m}[f(a) + 2\sum_{i=1}^{m-1} f(a_{2i}) + 4\sum_{i=0}^{m-1} f(a_{2i+1}) + f(b)] - \qquad (1.2)$$

$$-\frac{(b-a)^5}{2880m^4}f^{(4)}(\xi),$$

unde $a_i = a + i\frac{b-a}{2m},\ i \in \{0, 1, \ldots, 2m\}$.

Practic, integrala se aproximează prin

$$J_m = \frac{b-a}{6m}[f(a) + 2\sum_{i=1}^{m-1} f(a_{2i}) + 4\sum_{i=0}^{m-1} f(a_{2i+1}) + f(b)]$$

ultimul termen din (1.2), denumit rest, se neglijează. Astfel apare o eroare de metodă care se suprapune peste erorile de rotunjire datorate reprezentării numerelor reale în virgulă mobilă din memoria unui calculator şi calculului aritmetic corespunzător.

Schema adaptivă constă din calculul unui şir $(J_{m_k})_k$. Elementele şirului se calculează până la îndeplinirea unei condiţii din regula de oprire. Şirul $(m_k)_{k\in\mathbb{N}}$ il vom defini prin formula de recurenţă $m_{k+1} = 2m_k$.

În acest fel, dacă notăm prin S_m^p, S_m^i sumele

$$S_m^p = \sum_{i=1}^{m-1} f(a_{2i}) \quad \text{şi} \quad S_m^i = \sum_{i=0}^{m-1} f(a_{2i+1})$$

atunci $S_{m+1}^p = S_m^p + S_m^i$. La fiecare nouă iteraţie va trebui calculată doar suma S_{m+1}^i.

Datele de intrare pe care clientul trebuie să le furnizeze sunt:

- expresia de integrat $f(x)$;
- limitele de integrare;
- toleranţa;
- numărul maxim admis de iteraţii.

Datele de ieşire pe care le aşteptăm să le primim de la programul de calcul sunt:

- indicatorul de răspuns al programului;
- aproximaţia integralei;
- numărul iteraţiilor efectuate.

1.3 Probleme conexe

La elaborarea şi dezvoltarea programelor suntem confruntaţi cu problemele:

- *Jurnalizarea* adică afişarea / reţinerea rezultatelor sau evenimentelor într-un fişier. Deseori prezintă interes evoluţia procesului de calcul prin prisma unor rezultate intermediare. În acest sens se pot utiliza pachetele / produsele: `java.util.logging` din `jdk, apache-log4j` (*The Apache Software Foundation*) - `www.apache.org`, `slf4j` (*Simple Logging Facade for Java*) - `www.QOS.ch`, Quality of Open Software, `logback` - `logback.qos.ch`.

- Verificarea rezultatelor obţinute. Fiind dat rezultatul, fiecare dintre problemele propuse este o problemă de test. Rolul unei probleme de test este verificarea funcţionării programului de rezolvare, depistarea unor greşeli. În acest sens vom utiliza produsul *junit.*

  Alt produs cu acelaşi scop este *TestNG.*

- Dezvoltarea aplicaţiei, în sensul simplificării operaţiilor de compilare, arhivare, rularea problemelor de test, etc. Vom arăta modul de folosire pentru

  – *apache-ant.* Se presupune că toate resursele utilizate, cuprinse uzual în fişiere cu extensia `jar` (*j*ava *ar*hive) sunt disponibile pe calculatorul local. Dacă calculatorul este conectat la Internet, atunci, folosind suplimentar *apache-ivy,* resursele publice pot fi descărcate împreună cu toate dependinţele şi utilizate prin *apache-ant.*

  – *apache-maven*[1] este un alt cadru de dezvoltare şi gestiune a proiectelor (Project management framework). Calculatorul pe care se dezvoltă proiectul / aplicaţia trebuie să fie conectat la Internet. Resursele necesare îndeplinirii diferitelor sarcini (*maven artifacts*) sunt preluate din Internet şi depuse într-un depozit local *maven* (*local repository*). În prezent sunt întreţinute depozite publice de resurse soft necesare dezvoltării de aplicaţii cu *maven*[2] iar dezvoltatorii de instrumente soft au posibilitatea de a-şi promova produsele prin depunerea într-un depozit *maven.* Dintr-un asemenea depozit public resursele necesare sunt descărcate în depozitul local.

1.3.1 Jurnalizare

Jurnalizare prin `java.util.logging`

Şablonul de programare cu afişarea mesajelor pe ecranul monitorului este

```
1  import java.util.logging.Logger;

3  public class Exemplu{
4    static Logger logger = Logger.getLogger(Exemplu.class.getName());

6    public static void main(String args[]) {
7       logger.severe("SEVERE : Hello");
8       logger.warning("WARNING : Hello");
9       logger.info("INFO : Hello");
10   }
11 }
```

Programul afişează

```
Jan 23, 2013 2:34:40 PM Exemplu main
SEVERE: SEVERE : Hello
Jan 23, 2013 2:34:40 PM Exemplu main
WARNING: WARNING : Hello
Jan 23, 2013 2:34:40 PM Exemplu main
INFO: INFO : Hello
```

[1] Maven – acumulator de cunoştinţe (Idiş).

[2] De exemplu `repo1.maven.org/maven2`.

Dacă dorim ca rezultatele să fie înscrise într-un fişier, de exemplu *logging.txt* atunci clasa de mai sus se modifică în

```
1  import java.util.logging.Logger;
2  import java.util.logging.FileHandler;
3  import java.util.logging.SimpleFormatter;
4  import java.io.IOException;

6  public class Exemplu{
7    static Logger logger = Logger.getLogger(Exemplu.class.getName());

9    public static void main(String[] args) {
10     try{
11       FileHandler loggingFile = new FileHandler("logging.txt");
12       loggingFile.setFormatter(new SimpleFormatter());
13       logger.addHandler(loggingFile);
14     }
15     catch(IOException e){
16       System.out.println(e.getMessage());
17     }
18     logger.severe("SEVERE : Hello");
19     logger.warning("WARNING : Hello");
20     logger.info("INFO : Hello");
21   }
22 }
```

1.3.2 Verificarea rezultatelor cu *junit*

junit permite verificarea automată a rezultatelor furnizate de un program, pentru o mulţime de date de test.

Utilizarea produsului necesită declararea în variabila de sistem `classpath` a fişierelor *junit-*.jar, hamcrest-core-*.jar.*

Utilizarea produsului într-un program Java constă din:

1. Declararea resurselor pachetului *junit* prin
   `import org.junit.*;`
   `import static org.junit.Assert.*;`

2. Declararea clasei cu testele *junit* - uzual în metoda `main`.
   `org.junit.runner.JUnitCore.main("`*AppClass*`");`

3. Eventuale operaţii necesare înainte sau după efectuarea testelor se precizează respectiv, în câte o metodă care a fost declarată cu adnotarea `@org.junit.Before` şi respectiv, `@org.junit.After`.

4. Testele se definesc în metode declarate cu adnotarea `@org.junit.Test`.

   Clasa `Assert` posedă metodele de verificare ale unui rezultat:

   - `static void assertEquals(`Tip *aşteptat*, Tip *actual*`)`
     unde Tip poate fi `double, int, long, Object`.
   - `static void assertEquals(double` *aşteptat,* `double` *actual,* `double` *delta*`)`
     Testul reuşeşte dacă $|aşteptat\text{-}actual| < delta$.

- `static void assertArrayEquals(`Tip[ ] *aşteptat*, Tip[ ] *actual*`)`
  unde Tip poate fi `byte, char, int, long, short, Object.`
- `static void assertTrue(boolean` *condiţie*`)`
- `static void assertFalse(boolean` *condiţie*`)`
- `static void assertNull(Object` *object*`)`
- `static void assertNotNull(Object` *object*`)`

În cazul exemplului

```
1  import org.junit.*;
2  import static org.junit.Assert.*;

4  public class Exemplu{
5    public double rezultat=1.0;
6    public double eps=1e-6;

8    double getValue(){
9      return 1.0000001;
10   }

12   @Test
13   public void test(){
14     assertEquals(rezultat,getValue(),eps);
15   }

17   public static void main(String[] args){
18     org.junit.runner.JUnitCore.main("Exemplu");
19   }
20 }
```

se obţine

```
JUnit version 4.5
.
Time: 0.03

OK (1 test)
```

1.3.3 Unealta de dezvoltare *apache-ant*

Începem această secţiune prin a introduce câteva elemente privind sintaxa într-un document `xml` pentru că *apache-ant* face apel la un fişier `xml`.

XML

Extensible Markup Language (XML) reprezintă un limbaj pentru definirea marcajelor de semantică, care împart un document în părţi identificabile în document.

Totodată XML este un meta-limbaj pentru definirea sintaxei de utilizat în alte domenii.

XML descrie structura şi semantica şi nu formatarea.

Structura unui document `XML` este

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   corpul documentului alcatuit din elemente
```

Prima linie reprezintă declaraţia de document XML.

Corpul documentului este alcătuit din elemente. Începutul unui element este indicat printr-un marcaj. Textul marcajului constituie denumirea elementului. Elementele pot fi cu corp, alcătuit din alte elemente, având sintaxa

```
<marcaj>
    corpul elementului
</marcaj>
```

sau fără corp, caz în care sintaxa este

```
<marcaj/>
```

Un marcaj poate avea atribute date prin sintaxa

```
numeAtribut="valoareAtribut"
```

Valoarea unui atribut este cuprinsă între ghilimele (””).

```
<marcaj numeAtribut="valoareAtribut" . . .>
    corpul elementului
</marcaj>
```

Există un singur element rădăcină. Elementele unui document XML formează un arbore. Fiecărui marcaj de început al unui element trebuie să-i corespundă un marcaj de sfârşit. Caracterele mari şi mici din denumirea unui element sunt distincte (*case sensitive*).

Elementele încuibărite (*nested*)- incluse într-un alt element - nu se pot amesteca, adică un marcaj de sfârşit corespunde ultimului marcaj de început.

Un comentariu se indică prin

```
<!--
  Text comentariu
-->
```

Exemplul 1.3.1 *Fişier* `XML` *- denumirile elementelor şi conţinutul lor permit înţelegerea simplă a semanticii introduse în document.*

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <cursuri>
3    <disciplina fel="obligatoriu">
4      <nume> Analiza numerica </nume>
5      <fond-de-timp>
6         <curs> 2 </curs>
7         <seminar> 1 </seminar>
8         <laborator> 1 </laborator>
9      </fond-de-timp>
10   </disciplina>
11   <disciplina fel="obligatoriu">
12     <nume> Programare distribuita </nume>
13     <fond-de-timp>
14        <curs> 2 </curs>
15        <seminar> 0 </seminar>
```

```
16          <laborator> 2 </laborator>
17       </fond-de-timp>
18    </disciplina>
19    <disciplina fel="obligatoriu">
20       <nume> Soft matematic </nume>
21       <fond-de-timp>
22          <curs> 2 </curs>
23          <seminar> 0 </seminar>
24          <laborator> 1 </laborator>
25       </fond-de-timp>
26    </disciplina>
27 </cursuri>
```

Amintim faptul că reprezentarea obiectelor matematice prin elemente XML constituie subiectul a două proiecte *MathML* şi *OpenMath.* Obiectivul limbajului *MathML* este reprezentarea unui text matematic într-un document HTML, în timp ce obiectivul proiectului *OpenMath* este reprezentarea semantică a datelor matematice pentru realizarea de aplicaţii cooperante între sisteme de calcul simbolic - CAS (*Computer Algebra System*).

apache-ant

Utilitarul *apache-ant* asigură executarea unui şir de comenzi de operare. Aceste comenzi se înregistrează într-un fişier de tip `xml`, cu denumirea `build.xml`. Astfel, *apache-ant* se substituie unui fişier de comenzi `bat` în Windows sau unui script shell din Linux/Unix. Avantajul obţinut constă în independenţa faţă de platforma de calcul (Windows, Linux).

Instalarea constă în dezarhivarea fişierului descărcat din Internet.

Lansarea în execuţie necesită fixarea variabilei de mediu `JAVA_HOME`, ce conţine calea la distribuţia Java. Lansarea se poate face prin următorul fişier de comenzi

```
set JAVA_HOME=. . .
set ANT_HOME=. . .
%ANT_HOME%\bin\ant.bat %1
```

Parametrul %1 acestui fişier de comenzi reprezintă obiectivul care se doreşte a fi atins. Dacă se modifică denumirea sau locaţia fişierului `build.xml` atunci fişierul de comenzi se invocă cu opţiunea `-buildfile`.

Un fişier `build.xml` corespunde unui proiect (project), alcătuit din unul sau mai multe obiective (target). Atingerea fiecărui obiectiv constă din îndeplinirea uneia sau mai multor sarcini (task). *Apache-ant* conţine o familie predefinită de sarcini. Programatorul are datoria fixării atributelor sarcinilor. Manualul din documentaţia produsului conţine descrierea atributelor cât şi exemple. În general, o sarcină reprezintă o operaţie executată uzual în linia de comandă.

Atributele se dau, respectând sintaxa `XML`

$$\text{numeAtribut} = \text{"valoareAtribut"}$$

Astfel, un proiect apare sub forma

```
<project name="numeProiect"
         default="obiectiv"
         basedir="catalogDeReferinta">

<target name="numeObiectiv">
     sarcini
</target>
. . . . . .
</project>
```

Dacă la apelarea lui *Apache-ant* lipseşte parametrul opţional atunci se va executa obiectivul `default`.

Într-un proiect se pot defini variabile prin elementul

<property name="numeVariabila" value="valoareVariabila" />

O variabilă definită se va utiliza cu sintaxa ${numeVariabila}.

1.3.4 Gestiunea proiectelor cu *apache-maven*

Instalarea produsului constă în dezarhivarea fişierului descărcat din Internet într-un catalog a cărei cale este fixată în variabila de mediu `MAVEN_HOME`.

Utilizarea produsului necesită

- Completarea variabilei de sistem `PATH` cu calea `MAVEN_HOME\bin`.
- Declararea variabilei `JAVA_HOME` având ca valoare calea către distribuţia `jdk` folosită.

În mod obişnuit depozitul local *maven* este

`C:\Documents and Settings\`*client*`\.m2\repository`

Locaţia depozitului local se poate modifica, introducând elementul

```
<localRepository>volum:/cale/catalog_depozit</localRepository>
```

în fişierul `MAVEN_HOME\conf\settings.xml`.

Potrivit principiului separării preocupărilor (Separation of Conserns), un proiect *maven* produce o singură ieşire.

Declararea unui proiect se face printr-un fişier `pom.xml` (Project Object Model). Este sarcina programatorului să completeze fişierul `pom.xml`, creat la generarea structurii de cataloage ale proiectului, cu specificarea resurselor suplimentare sau a condiţionărilor în efectuarea unor operaţii (de exemplu, prezenţa adnotărilor necesită utilizarea unei versiuni Java mai mare dacât 1.5).

Dezvoltarea unei aplicaţii / proiect prin *maven* presupune generarea unei structuri de cataloage (Standard directory layout for projects). Această structură de cataloage este specifică tipului / şablonului de aplicaţie (*archetype,* în limbajul *maven*).

Şabloane uzuale de aplicaţii:

Nume şablon	Semnificaţia
`maven-archetype-quickstart`	aplicaţie simplă (şablonul implicit)
`maven-archetype-webapp`	aplicaţie Web

Îndeplinirea diferitelor obiective (generarea unui proiect, compilare, arhivare, testare, etc) se obţin prin comenzi *maven*.

Comenzile *maven* sunt de două tipuri:

- Comenzi pentru gestiunea ciclului de viaţă al unui proiect (lifecycle commands):

Comanda *maven*	Semnificaţia
`mvn -version`	afişează versiunea lui *maven* (utilă pentru verificarea funcţionării lui *maven*)
`mvn clean`	şterge fişierele *maven* generate
`mvn compile`	compilează sursele Java
`mvn test-compile`	compilează sursele Java care realizează testele `junit`
`mvn test`	execută testul `junit`
`mvn package`	crează o arhivă jar sau war
`mvn install`	depune arhiva jar sau war în depozitul local

- Comenzi de operare inserate (plugin commands):

Comanda *maven*	Semnificaţia
`mvn -B archetype:generate`	generează structura de cataloage a proiectului Opţiunea -B are ca efect generarea neinteractivă. `mvn -B archetype:generate` \ `-DgroupId=`*numelePachetuluiAplicaţiei* \ `-DartifactId=`*numeleProiectului* \ `-DarchetypeArtifactId=`*numeŞablon*\ `-Dversion=`*versiuneaProiectului*
`mvn clean:clean`	şterge fişierele generate în urma compilării
`mvn compiler:compile`	compilează sursele Java
`mvn surefire:test`	execută testul `junit`
`mvn jar:jar`	crează o arhivă jar
`mvn install:install-file`	depune o arhivă jar în depozitul local `mvn install:install-file` \ `-Dfile=`*numeFişier* \ `-DgroupId=`*numePachet* \ `-DartifactId=`*numeProiect* \ `-Dversion=`*versiunea* \ `-Dpackaging=`*tipArhivă*
`mvn exec:java`	execută metoda `main` a unei clase `mvn exec:java` \ `-Dexec.mainClass=`*"clasaMetodeiMain"*
`mvn dependency:copy-dependencies`	descarcă în catalogul `target` resursele declarate în `dependencies`.

Astfel comanda

```
mvn -B archetype:generate -DgroupId=unitbv.cs.calcul -DartifactId=hello -Dversion=1.0
```

generează arborescenţa

```
hello
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> unitbv
|    |    |    |    |--> cs
|    |    |    |    |    |--> calcul
|    |    |    |    |    |    |  App.java
|    |--> test
|    |    |--> java
|    |    |    |--> unitbv
|    |    |    |    |--> cs
|    |    |    |    |    |--> calcul
|    |    |    |    |    |    |   AppTest.java
|    pom.xml
```

Descrierea proiectului este dată în fişierul `pom.xml` generat

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
```

```
4    http://maven.apache.org/maven-v4_0_0.xsd">
5   <modelVersion>4.0.0</modelVersion>
6   <groupId>unitbv.cs.calcul</groupId>
7   <artifactId>hello</artifactId>
8   <packaging>jar</packaging>
9   <version>1.0</version>
10  <name>hello</name>
11  <url>http://maven.apache.org</url>
12  <dependencies>
13    <dependency>
14      <groupId>junit</groupId>
15      <artifactId>junit</artifactId>
16      <version>3.8.1</version>
17      <scope>test</scope>
18    </dependency>
19  </dependencies>
20 </project>
```

App.java este programul Java *HelloWorld* iar *AppTest.java* este un program de verificare bazat pe `junit`.

Pentru testarea aplicaţiei, din catalogul *hello*, se execută comenzile

```
mvn compile
mvn test
```

Execuţia programului se poate lansa prin intermediul unui profil (*profile*)

```
mvn exec:java -PnumeProfil
```

În prealabil în fişierul `pom.xml` se introduce secvenţa

```
<profiles>
    <profile>
      <id>numeProfil</id>
      <properties>
        <target.main.class>clasaCuMetodaMain</target.main.class>
      </properties>
    </profile>
  </profiles>

  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <configuration>
          <mainClass>${target.main.class}</mainClass>
          <includePluginDependencies>false</includePluginDependencies>
        </configuration>
      </plugin>
    </plugins>
  </build>
```

Această variantă este avantajoasă în cazul în care proiectul include mai multe clase cu metoda `main`.

Sarcina programatorul este acela de a înlocui aceste programe cu cele care rezolvă sarcinile proiectului. Pentru orice prelucrare toate dependinţele trebuie să se găsească în depozitul local *maven*. Dacă o dependinţă (resursă `jar`) nu se găseşte în depozitul local atunci resursa este căutată într-un depozit global şi este descărcată în depozitul global. Este sarcina programatorului să declare toate dependinţele necesare unei aplicaţii. Declararea se face într-un element <`dependency`>. Dacă resursa este inaccesibilă atunci *maven* termină prelucrarea.

Programatorul are posibilitatea să specifice depozite globale unde să se găsească resursele necesare, de exemplu

```
<repositories>
   <repository>
      <id>java.net-promoted</id>
      <url>https://maven.java.net/content/groups/promoted/</url>
   </repository>
</repositories>
```

Se pot defini variabile în elementul

```
<properties>
   <nume.proprietate>valoare</nume.proprietate>
   . . .
</properties>
```

O variabilă se indică la fel ca în *apache-ant,* prin *${nume.proprietate}*.

În *maven* se pot integra sarcini *apache-ant* pentru orice etapă al evoluţiei unei aplicaţii. Utilizarea constă în completarea fişierului `pom.xml` cu

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-antrun-plugin</artifactId>
      <executions>
        <execution>
          <phase>
             <!-- etapa de viata : compile, package, install, test -->
          </phase>
          <configuration>
            <tasks>
              <!-- Exemplu
              <property name="compile_classpath" refid="maven.compile.classpath"/>
              <property name="runtime_classpath" refid="maven.runtime.classpath"/>
              <property name="test_classpath" refid="maven.test.classpath"/>
              <property name="plugin_classpath" refid="maven.plugin.classpath"/>

              <echo message="compile classpath: ${compile_classpath}"/>
              <echo message="runtime classpath: ${runtime_classpath}"/>
              <echo message="test classpath:    ${test_classpath}"/>
              <echo message="plugin classpath:  ${plugin_classpath}"/>
              -->
            </tasks>
          </configuration>
          <goals>
            <goal>run</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

În elementul <`tasks`> se definesc sarcinile `ant` care se doresc executate la comanda corespunzătoare prelucrării etapei din evoluţia proiectului *maven* - compile, package, install, test, etc.

1.4 O mini-bibliotecă numerică

Programele corespunzătoare metodelor de calcul (metoda tangentei şi metoda Simpson) le vom include într-o mini-bibliotecă. Această bibliotecă va fi valorificată în capitolele următoare, prin dezvoltarea de programe client cu interfeţe grafice prietenoase şi prin apelarea resurselor mini-bibliotecii de la distanţă.

În timpul programării apare problema transmiterii datelor către câmpurile claselor sau către parametrii metodelor. Probleme apar şi în cazul rezultatelor returnate de metode. Deseori printre date se află expresii ale unor funcţii matematice.

Semnalăm următoarele posibilităţi de programare a transmiterii datelor:

- Datele - mai puţin funcţiile - se transmit ca şi parametri iar o funcţie se poate defini într-o clasă care eventual implementează o interfaţă predefinită. O instanţă a unei asemenea clase se transmite de asemenea ca parametru. Această soluţie corespunde modului de programare din Fortran sau C [7].

- Prin clase acoperitoare a datelor şi respectiv, a rezultatelor.

Varianta în care o funcţie este dată prin expresia ei de calcul fixată printr-un şir de caractere (string) va fi tratată în Cap. 5.

În continuare, varianta adoptată va fi cea în care datele se transmit prin clase acoperitoare. Avantajul acestei soluţii este dat de flexibilitatea oferită de paradigma programării orientate pe obiecte, iar dezavantajul constă în cerinţa ca un utilizator (client) să aibă cunoştinţe de programare orientată pe obiecte.

Pentru fiecare tip de problemă, mini-biblioteca va conţine:

- o interfaţă în care se declară metodele disponibile;

- o implementare a interfeţei.

Astfel, se oferă posibilitatea dezvoltării şi utilizării mai multor implementări ale aceleiaşi metode.

Clasele Java prin care se transmitit datele de intrare (*DataIn*) şi cele de ieşire (*DataOut*) sunt incluse în pachetele interfeţelor corespunzătoare. Aceste clase sunt componente Java (*bean*).

Desfăşurarea mini-bibliotecii *mathlib* cu programele sursă este

```
lib
|   log4j-*.jar
src
|--> mathlib
|    |--> client
|    |    |--> ecalg
|    |    |    |--> impl
|    |    |    |    |   MetodaTangentei.java // implementeaza interfata
|    |    |    |    IMetodaTangentei.java    // interfata
|    |    |    |    DataIn.java
|    |    |    |    DataOut.java
|    |    |--> cvadra
|    |    |    |--> impl
|    |    |    |    |   MetodaSimpson.java  // implementeaza interfata
|    |    |    |    IMetodaSimpson.java     // interfata
|    |    |    |    DataIn.java
|    |    |    |    DataOut.java
```

Utilizarea mini-bibliotecii ca un modul *Google Web Toolkit* impune intercalarea catalogului *client.*

Codurile claselor / interfeţelor sunt:

1. Clasa *mathlib.client.ecalg.IMetodaTangentei*

```
1  package mathlib.client.ecalg;
2  /**
3   * Interfata metodei tangentei.
4   */
5  public interface IMetodaTangentei{
6    /**
7     * Metoda tangentei.
8     */
9    public DataOut metodaTangentei(DataIn din);
10 }
```

2. Clasa *mathlib.client.ecalg.DataIn*

```
1  package mathlib.client.ecalg;
2  /**
3   *  Clasa acoperitoare a datelor necesare rezolvarii
4   *  unei ecuatii algebrice
5   */
6  public abstract class DataIn{
7    private double x;   // aproximatia initiala
8    private double eps; // toleranta admisa
9    private int nmi;       // numar maxim admis de iteratii

11   /**
12    *  Functia corespunzatioare membrului stang al ecuatiei fct(x)=0.
13    */
14   public abstract double fct(double x);

16   /**
17    *  Fixeaza aproximatia solutiei.
18    */
19   public void setX(double x){
20     this.x=x;
21   }
22   /**
23    *  Fixeaza extremitatea inferioara a intervalului de integrare
24    */
25   public double getX(){
26     return x;
27   }

29   /**
30    *  Fixeaza toleranta.
31    */
32   public void setEps(double eps){
33     this.eps=eps;
34   }
35   /**
36    *  Returneaza toleranta.
37    */
38   public double getEps(){
39     return eps;
40   }

42   /**
43    *  Fixeaza numarul maxim admis de iteratii.
44    */
45   public void setNmi(int nmi){
46     this.nmi=nmi;
```

```
47    }
48    /**
49     *  Returneaza numarul maxim admis de iteratii.
50     */
51    public int getNmi(){
52      return nmi;
53    }
54  }
```

3. Clasa *mathlib.client.ecalg.DataOut*

```
1   package mathlib.client.ecalg;
2   /**
3    *  Clasa acoperitoare a rezultatelor obtinute
4    *  la rezolvarea unei ecuatii algebrice
5    */
6   public class DataOut {
7     private double x;    // aproximatia calculata a solutiei
8     private int ind;     // indicatorul de raspuns al programului
9     private double f;    // valoarea expresiei stangi a ecuatiei in x
10    private int ni;      // numarul iteratiilor efectuate

12    /**
13     *  Returneaza aproximatia solutiei.
14     */
15    public double getX(){
16      return x;
17    }
18    /**
19     *  Fixeaza aproximatia solutiei.
20     */
21    public void setX(double x){
22      this.x=x;
23    }

25    /**
26     *  Returneaza valoarea membrului stang al ecuatiei calculata
27     *  in aproximatia fixata a solutiei ecuatiei.
28     */
29    public double getF(){
30      return f;
31    }
32    /**
33     *  Fixeaza valoarea membrului stang al ecuatiei calculata
34     *  in aproximatia calculata a solutiei ecuatiei.
35     */
36    public void setF(double f){
37      this.f=f;
38    }

40    /**
41     *  Returneaza indicatorul de raspuns.
42     *
43     *  0 - succes;
44     *  1 - insucces.
45     *  2 - singularitate depistata in timpul calculului.
46     */
47    public int getInd(){
48      return ind;
49    }
50    /**
51     *  Fixeaza indicatorul de raspuns.
52     */
53    public void setInd(int ind){
54      this.ind=ind;
55    }
```

```
57    /**
58     *  Returneaza numarul iteratiilor efectuate.
59     */
60    public int getNi(){
61      return ni;
62    }
63    /**
64     *  Fixeaza numarul iteratiilor efectuate.
65     */
66    public void setNi(int ni){
67      this.ni=ni;
68    }
69  }
```

4. Clasa *mathlib.client.ecalg.impl.MetodaTangentei*

```
1  package mathlib.client.ecalg.impl;
2  import java.util.logging.Logger;
3  import java.util.logging.FileHandler;
4  import java.util.logging.SimpleFormatter;
5  import mathlib.client.ecalg.DataIn;
6  import mathlib.client.ecalg.DataOut;
7  import mathlib.client.ecalg.IMetodaTangentei;
8  import java.io.IOException;

10 /**
11  * Implementarea metodei tangentei.
12  * Varianta cu inregistrarea rezultatelor intermediare (log).
13  */
14 public class MetodaTangentei implements IMetodaTangentei{
15   static Logger logger = Logger.getLogger(MetodaTangentei.class.getName());

17   public MetodaTangentei(){
18     try{
19       FileHandler loggingFile = new FileHandler("resultsEcalg.log");
20       loggingFile.setFormatter(new SimpleFormatter());
21       logger.addHandler(loggingFile);
22     }
23     catch(IOException e){
24       System.out.println(e.getMessage());
25     }
26   }

28   /**
29    * Metoda tangentei aplicata datelor fixate in obiectul din.
30    */
31   public DataOut metodaTangentei(DataIn din){
32     double x,y=din.getX(),d,f,df,eps=din.getEps();
33     int nmi=din.getNmi();
34     DataOut dout=new DataOut();
35     int ni=0;
36     do{
37       ni++;
38       x=y;
39       f=din.fct(x);
40       df=dfct(x,din);
41       y=x-f/df;
42       if((y==Double.NaN)||(Math.abs(y)==Double.POSITIVE_INFINITY)){
43         dout.setInd(2);
44         dout.setX(y);
45         dout.setF(Double.NaN);
46         dout.setNi(ni);
47         return dout;
48       }
49       d=Math.abs((y-x));
```

```
50        String mesaj="iter = "+ni+" solutia = "+x+" eroarea = "+d;
51        logger.info(mesaj);
52      }
53      while((d>=eps) && (ni<nmi));
54      if(d<eps)
55        dout.setInd(0);
56      else
57        dout.setInd(1);
58      dout.setX(x);
59      dout.setF(din.fct(x));
60      dout.setNi(ni);
61      return dout;
62    }

64    /**
65     * Calculul derivatei de ordinul intai.
66     * @param x Punctul in care se calculeaza derivata.
67     * @param din Obiect DataIn care contine definitia functiei a carei derivata
68     * se calculeaza.
69     */
70    private double dfct(double x, DataIn din){
71      int m=6;
72      double h=1e-10;
73      double pk=1; // 4^k
74      double[][] d=new double[m][m];
75      for(int i=0;i<m;i++)
76        d[i][0]=0.5*(din.fct(x+h)-din.fct(x-h))/h;
77      for(int j=1;j<m;j++){
78        pk=4*pk;
79        for(int i=j;i<m;i++){
80          d[i][j]=(pk*d[i][j-1]-d[i-1][j-1])/(pk-1);
81        }
82      }
83      return d[m-1][m-1];
84    }
85  }
```

5. Clasa *mathlib.client.cvadra.IMetodaSimpson*

```
1  package mathlib.client.cvadra;
2  /**
3   * Interfata metodei Simpson.
4   */
5  public interface IMetodaSimpson{
6    /**
7     * Metoda Simpson.
8     */
9    public DataOut metodaSimpson(DataIn din);
10 }
```

6. Clasa *mathlib.client.cvadra.DataIn*

```
1  package mathlib.client.cvadra;
2  /**
3   *  Clasa acoperitoare a datelor necesare calculului
4   *  unei integrale.
5   */
6  public abstract class DataIn{

8    private double a;        // extremitatea stanga
9    private double b;        // extremitatea dreapta
10   private double eps;      // toleranta admisa
11   private int nmi;         // numar maxim admis de iteratii

13   /**
```

```
14      *  Functia integrata.
15      */
16     public abstract double fct(double x);

18     /**
19      *  Returneaza extremitatea inferioara a intervalului de integrare.
20      */
21     public double getA(){
22       return a;
23     }
24     /**
25      *  Fixeaza extremitatea inferioara a intervalului de integrare.
26      */
27     public void setA(double a){
28       this.a=a;
29     }

31     /**
32      *  Returneaza extremitatea superioara a intervalului de integrare.
33      */
34     public double getB(){
35       return b;
36     }
37     /**
38      *  Fixeaza extremitatea superiora a intervalului de integrare.
39      */
40     public void setB(double b){
41       this.b=b;
42     }

44     /**
45      *  Fixeza toleranta.
46      */
47     public void setEps(double eps){
48       this.eps=eps;
49     }
50     /**
51      *  Returneaza toleranta.
52      */
53     public double getEps(){
54       return eps;
55     }

57     /**
58      *  Fixeaza numarul maxim admis de iteratii.
59      */
60     public void setNmi(int nmi){
61       this.nmi=nmi;
62     }
63     /**
64      *  Returneaza numarul maxim admis de iteratii.
65      */
66     public int getNmi(){
67       return nmi;
68     }
69   }
```

7. Clasa *mathlib.client.cvadra.DataOut*

```
1  package mathlib.client.cvadra;
2  /**
3   *  Clasa acoperitoare a rezultatelor obtinute
4   *  la calculul unei integrale.
5   */
6  public class DataOut{
7    private double integrala; // integrala
```

```
8     private int ind;           // indicatorul de raspuns al programului
9     private int ni;            // numarul iteratiilor efectuate

11    /**
12     *  Returneaza valoarea integralei.
13     */
14    public double getIntegrala(){
15      return integrala;
16    }
17    /**
18     *  Fixeaza valoarea integralei.
19     */
20    public void setIntegrala(double integrala){
21      this.integrala=integrala;
22    }

24    /**
25     *  Returneaza indicatorul de raspuns.
26     *
27     *  0 - succes;
28     *  1 - insucces.
29     */
30    public int getInd(){
31      return ind;
32    }
33    /**
34     *  Fixeaza indicatorul de raspuns.
35     */
36    public void setInd(int ind){
37      this.ind=ind;
38    }

40    /**
41     *  Returneaza numarul iteratiilor efectuate.
42     */
43    public int getNi(){
44      return ni;
45    }
46    /**
47     *  Fixeaza numarul iteratiilor efectuate.
48     */
49    public void setNi(int ni){
50      this.ni=ni;
51    }
52  }
```

8. Clasa *mathlib.client.cvadra.impl.MetodaSimpson*

```
1  package mathlib.client.cvadra.impl;
2  import java.util.logging.Logger;
3  import java.util.logging.FileHandler;
4  import java.util.logging.SimpleFormatter;
5  import mathlib.client.cvadra.DataIn;
6  import mathlib.client.cvadra.DataOut;
7  import mathlib.client.cvadra.IMetodaSimpson;
8  import java.io.IOException;
9  /**
10  * Implementarea metodei Simpson.
11  * Varianta cu inregistrarea rezultatelor intermediare (log).
12  */
13 public class MetodaSimpson implements IMetodaSimpson{
14   private double s=Double.NaN;
15   static Logger logger = Logger.getLogger(MetodaSimpson.class.getName());

17   public MetodaSimpson(){
18     try{
```

```
19        FileHandler loggingFile = new FileHandler("resultsInteg.log");
20        loggingFile.setFormatter(new SimpleFormatter());
21        logger.addHandler(loggingFile);
22      }
23      catch(IOException e){
24        System.out.println(e.getMessage());
25      }
26    }

28    /*
29      * Formula Simpson de aplicare practica cu
30      * parametru de discretizare fixat.
31      * La prima apelare s=Double.NaN !
32    */
33    private double simpson(int m,DataIn din){
34      double a=din.getA(),b=din.getB();
35      double h=0.5*(b-a)/m;
36      double sp=0,si=0;
37      if(Double.isNaN(s)){
38        for(int i=1;i<m;i++)
39          sp+=din.fct(a+2*i*h);
40      }
41      else
42        sp=s;
43      for(int i=0;i<m;i++)
44        si+=din.fct(a+(2*i+1)*h);
45      s=sp+si;
46      return h*(din.fct(a)+din.fct(b)+2*sp+4*si)/3;
47    }

49    /**
50     * Schema adaptiva pentru metoda lui Simpson.
51     * Algoritm iterativ cu regula de oprire.
52     */
53    public DataOut metodaSimpson(DataIn din){
54      double intv,intn,d,eps=din.getEps();
55      int nmi=din.getNmi();
56      int m=2;        // Parametrul de discretizare
57      intn=simpson(m,din);
58      DataOut dout=new DataOut();
59      int ni=0;
60      do{
61        ni++;
62        intv=intn;
63        m=2*m;
64        intn=simpson(m,din);
65        d=Math.abs(intn-intv);
66        String mesaj="iter = "+ni+" integrala = "+intn+" eroarea = "+d;
67        logger.info(mesaj);
68      }
69      while((d>=eps) && (ni<nmi));
70      if(d<eps)
71        dout.setInd(0);
72      else
73        dout.setInd(1);
74      dout.setIntegrala(intn);
75      dout.setNi(ni);
76      return dout;
77    }
78  }
```

Această mini-bibliotecă va fi extinsă pe parcursul lucrării, adăugând clase noi.

Compilarea şi arhivarea claselor se realizează prin intermediul lui *apache-ant* cu următorul fişier *build.xml*:

```
<?xml version='1.0' encoding='UTF-8'?>
<project name="proiect" basedir="." default="ToJar" >

  <property name="package" value="mathlib" />

  <property name="lib.dir" value="lib"/>
  <property name="build.dir" value="build"/>
  <property name="dist.dir" value="dist"/>
  <property name="dist.name" value="mathlib"/>

  <path id="pathref">
     <fileset dir="${lib.dir}">
       <include name="*.jar"/>
     </fileset>
     <pathelement path="${build.dir}"/>
  </path>

  <target name="Init">
    <mkdir dir="${build.dir}"/>
    <delete dir="${dist.dir}"/>
    <mkdir dir="${dist.dir}"/>
  </target>

  <target name="Compile" depends="Init" description="compile" >
     <javac srcdir="src/${package}"
           destdir="${build.dir}"
           includeantruntime="false">
        <classpath refid="pathref" />
     </javac>
  </target>

  <target name="ToJar" depends="Compile">
    <jar destfile="${dist.dir}/${dist.name}.jar" basedir="${build.dir}"/>
  </target>

  <target name="Docs">
    <delete dir="docs" />
    <mkdir dir="docs" />
    <javadoc destdir="docs" sourcepath="src" />
  </target>
</project>
```

Arhiva *mathlib.jar* care rezultă, va fi utilizată de programele *client.* În urma oricărei modificări efectuată în mini-bibliotecă trebuie recreată arhiva *mathlib.jar.*

Ori de câte ori se folosesc resursele mini-bibliotecii, variabila de sistem `classpath` trebuie să localizeze fişierul *mathlib.jar.*

1.5 Formatarea rezultatetor

Este un deziderat ca afişarea valorii unei variabile - de obicei un număr reprezentat în virgulă mobilă - să conţină doar cifrele ei semnificative, adică cele care nu sunt influenţate de erorile de rotunjire, care apar în urma reprezentării şi calculelor în virgulă mobilă, dar şi de eroarea de metodă.

Afişarea valorii variabilei - de exemplu r - prin `System.out.println(r)` se face cu un număr de zecimale, iar de la o anumită poziţie cifrele sunt influenţate de erorile de rotunjire şi de roarea de metodă.

Astfel apare cerinţa ca programatorul să specifice numărul de zecimale care să fie afişat, cu alte cuvinte să formateze rezultatul.

Atunci când afişarea are loc de un program *desktop,* formatarea se face cu metoda `printf` a clasei `PrintWriter.` Formate uzuale sunt $\%m.nX$ unde

- m - este numărul caracterelor;
- n - este numărul zecimalelor;
- $X \in \{e, f, g\}$; e - reprezentare ştiinţifică; f - reprezentare zecimală; g combinaţie între reprezentare ştiinţifică şi zecimală.

Dacă rezultatul apare prin intermediul unui program navigator, atunci se pot utiliza

- şablonul

```
import java.text.DecimalFormat;
. . .
DecimalFormat df=new DecimalFormat("0.00001");
out.println(df.format(r));
. . .
```

  unde *out* este fluxul de ieşire.

- metoda `format(String` *format,*`Object` *... args*`)` a clasei `java.io.PrintWriter.` O descriere amănunţită a modului de formatare este dată în documentaţia ce însoţeşte distribuţia jdk. Prezentăm doar cazul unei variabile numerice r de tip `float` sau `double` care se afişează pe $l = 12$ poziţii cu $d = 6$ cifre zecimale. Codul Java este

```
import java.io.PrintWriter;
. . .
PrintWriter out=. . .
out.format("r = %1$12.6f",r);
. . .
```

1.6 Rezolvarea problemelor

Deoarece clasele de tip *DataIn* ale mini-bibliotecii nu conţin codul funcţiei specifice unei probleme concrete, ele au fost declarate clase `abstract.` Un client trebuie să extindă o asemenea clasă specificând metoda *double fct(double x).*

Calculul soluţiei negative a ecuaţiei $2^x - x^2 = 0$

Aplicaţia client conţine clasele *SimpluEcAlgDataIn* - cu fixarea funcţiei - şi *TestEcAlg* - cu metoda `main` - având respectiv codurile

```
1  import mathlib.client.ecalg.DataIn;
2  public class SimpluEcAlgDataIn extends DataIn{
3
4    // membrul stang al ecuatiei
5    SimpluEcAlgDataIn(){
6      setX(-0.5);      // aproximatia initiala
7      setEps(1e-8);    // toleranta admisa
8      setNmi(50);      // numar maxim admis de iteratii
9    }
10
11   public double fct(double x){
12     return Math.exp(x*Math.log(2))-x*x;
13   }
14 }
```

şi

```
import org.junit.*;
import static org.junit.Assert.*;
import mathlib.client.ecalg.*;
import mathlib.client.ecalg.impl.*;

public class TestEcAlg{
  // rezultatul cunoscut al ecuatiei
  private double rezultat=-0.7666647;
  private DataIn din;
  private DataOut dout;

  @Before
  public void initializare(){
    din=new SimpluEcAlgDataIn();
    IMetodaTangentei obj=new MetodaTangentei();
    dout=obj.metodaTangentei(din);
  }

  @Test
  public void test(){
    assertEquals(rezultat,dout.getX(),din.getEps());
  }

  @After
  public void afisare(){
    System.out.println("\nIndicatorul de raspuns : "+dout.getInd());
    System.out.printf("Solutia ecuatiei : %16.8f\n",dout.getX());
    System.out.printf("Valoarea functiei in solutie : %16.8e\n",dout.getF());
    System.out.println("Numarul iteratiilor efectuate : "+dout.getNi());
  }

  public static void main(String[] args){
    org.junit.runner.JUnitCore.main("TestEcAlg");
  }
}
```

Se utilizează *junit* pentru verificarea rezultatului furnizat de aplicaţie cu soluţia cunoscută. Rezultatele afişate sunt

```
Indicatorul de raspuns : 0
Solutia ecuatiei :     -0.76666470
Valoarea functiei in solutie :  -5.14033260e-14
Numarul iteratiilor efectuate : 5

Time: 0.13
OK (1 test)
```

Calculul integralei $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x$

În mod asemănător aplicaţia client este alcătuită din clasele

```
import mathlib.client.cvadra.DataIn;
public class SimpluCvadraDataIn extends DataIn{
  SimpluCvadraDataIn(){
    setA(0);                // extremitatea stanga
    setB(0.25*Math.PI);     // extremitatea dreapta
    setEps(1e-12);          // toleranta admisa
    setNmi(50);             // numar maxim admis de iteratii
  }

  // functia de integrat
  public double fct(double x){
```

```
12      return Math.log(1+Math.tan(x));
13    }
14  }
```

şi

```
1  import org.junit.*;
2  import static org.junit.Assert.*;
3  import mathlib.client.cvadra.*;
4  import mathlib.client.cvadra.impl.*;

6  public class TestIntegrala{
7    // rezultatul cunoscut al integralei
8    private double rezultat=0.125*Math.PI*Math.log(2);
9    private DataIn din;
10   private DataOut dout;

12   @Before
13   public void initializare(){
14     din=new SimpluCvadraDataIn();
15     IMetodaSimpson obj=new MetodaSimpson();
16     dout=obj.metodaSimpson(din);
17   }

19   @Test
20   public void test(){
21     assertEquals(rezultat,dout.getIntegrala(),din.getEps());
22   }

24   @After
25   public void afisare(){
26     System.out.println("\nIndicatorul de raspuns : "+dout.getInd());
27     System.out.printf("Integrala : %16.8f\n",dout.getIntegrala());
28     System.out.println("Numarul iteratiilor efectuate : "+dout.getNi());
29   }

31   public static void main(String[] args){
32     org.junit.runner.JUnitCore.main("TestIntegrala");
33   }
34 }
```

Rezultatele obţinute sunt

```
Indicatorul de raspuns : 0
Integrala :      0.27219826
Numarul iteratiilor efectuate : 1

Time: 0.1
OK (1 test)
```

1.7 Dezvoltarea mini-bibliotecii cu *apache-maven*

Arătăm o modalitate de dezvoltare a mini-bibliotecii *mathlib* prin *apache-maven.* Deoarece un proiect *maven* produce o singură ieşire, este nevoie de 4 proiecte *maven*: două corespunzătoare celor două interfeţe (cu ieşirile *mathlib.client.ecalg, mathlib.client.cvadra*) şi alte două pentru implementarea lor (cu ieşirile *mathlib.client.ecalg.impl, mathlib.client.cvadra.impl*).

Clasele dezvoltate în §1.4 se utilizează nemodificate.

În prealabil arhivele *jep-2.4.1.jar* şi *matheclipse-parser-0.0.10.jar* pe care le utilizăm trebuie depuse în depozitul local

```
set HOME=locatia_curenta
start mvn install:install-file
  -Dfile=%HOME%\jep-2.4.1.jar
  -DgroupId=jep
  -DartifactId=jep
  -Dversion=2.4.1
  -Dpackaging=jar
start mvn install:install-file
  -Dfile=%HOME%\matheclipse-parser-0.0.10.jar
  -DgroupId=org.matheclipse
  -DartifactId=matheclipse-parser
  -Dversion=0.0.10
  -Dpackaging=jar
```

Proiectul interfaţei pentru rezolvarea unei ecuaţii algebrice se generează prin

```
set GroupId=mathlib.client.ecalg
set ArtifactId=iecalg
set Version=1.0
set ArchetypeArtifactId=maven-archetype-quickstart
mvn -B archetype:generate
  -DgroupId=%GroupId%
  -DartifactId=%ArtifactId%
  -Dversion=%Version%
  -DarchetypeArtifactId=%ArchetypeArtifactId%
```

După inserarea fişierelor java, desfăşurarea aplicaţiei devine

```
iecalg
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> mathlib
|    |    |    |    |--> client
|    |    |    |    |    |--> ecalg
|    |    |    |    |    |    |   DataIn.java
|    |    |    |    |    |    |   DataOut.java
|    |    |    |    |    |    |   IMetodaTangentei.java
|    pom.xml
```

Prelucrările necesare sunt: `mvn clean install` care înglobează ştergerea fişierelor generate anterior de *maven*, compilarea, arhivarea şi depunerea proiectului în depozitul local *maven*.

Generăm proiectul pentru implementarea interfeţei definită mai sus.

```
set GroupId=mathlib.client.ecalg.impl
set ArtifactId=ecalgimpl
set Version=1.0
set ArchetypeArtifactId=maven-archetype-quickstart
mvn -B archetype:generate
  -DgroupId=%GroupId%
  -DartifactId=%ArtifactId%
  -Dversion=%Version%
  -DarchetypeArtifactId=%ArchetypeArtifactId%
```

Partea de test se completează cu clasele utilizate în §1.5 pentru calculul soluţiei negative a ecuaţiei $2^x - x^2 = 0$. În final, desfăşurarea proiectului este

```
ecalgimpl
|--> src
|    |--> main
|    |    |--> java
|    |    |    |--> mathlib
|    |    |    |    |--> client
|    |    |    |    |    |--> ecalg
```

```
|    |    |    |    |    |    |--> impl
|    |    |    |    |    |    |    |   MetodaTangentei.java
|    |--> test
|    |    |--> java
|    |    |    |--> mathlib
|    |    |    |    |--> client
|    |    |    |    |    |--> ecalg
|    |    |    |    |    |    |--> impl
|    |    |    |    |    |    |    SimpluEcAlgDataIn.java
|    |    |    |    |    |    |    TestEcAlg.java
|    pom.xml
```

Fişierul `pom.xml` trebuie completat cu dependinţa pentru *iecalg*. Codul fişierului `pom.xml` devine

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>mathlib.client.ecalg.impl</groupId>
  <artifactId>ecalgimpl</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>ecalgimpl</name>
  <url>http://maven.apache.org</url>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>mathlib.client.ecalg</groupId>
      <artifactId>iecalg</artifactId>
      <version>1.0</version>
    </dependency>
  </dependencies>
</project>
```

Se pot face următoarele prelucrări: `mvn clean test`

Partea referitoare la formula de integrare numerică se tratează analog.

Odată dezvoltate, cele 4 proiecte pot fi asamblate într-un singur proiect *maven*, având structura

```
mathlib
|--> iecalg
|    |--> . . .
|--> ecalgimpl
|    |--> . . .
|--> icvadra
|    |--> . . .
|--> cvadraimpl
|    |--> . . .
|    pom.xml
```

cu fişierul `pom.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/maven-v4_0_0.xsd">

  <modelVersion>4.0.0</modelVersion>
```

```
8   <name> Maven mathlib </name>
9   <groupId>mathlib</groupId>
10  <version>1.0</version>
11  <artifactId>mathlib</artifactId>
12  <packaging>pom</packaging>

14  <modules>
15      <module>iecalg</module>
16      <module>ecalgimpl</module>
17      <module>icvadra</module>
18      <module>cvadraimpl</module>
19  </modules>
20 </project>
```

Oricare din comenzile *maven* va acţiona asupra fiecărui modul (modulul este denumirea dată unui proiect inclus).

Compilarea şi testarea se obţine cu comanda `mvn clean test`.

Compilarea, testarea, arhivarea şi copierea în depozitul local se obţine cu comanda `mvn clean install`.

1.8 Utilizare prin *OSGi*

OSGi - Open Sourse Gateway initiative, 1999, (semnificaţia numelui fiind astăzi depăşită) a dezvoltat un model de cadru de lucru privind:

- gestiunea ciclului de viaţă a unei aplicaţii (application life cycle management);
- registru de servicii;
- mediu de execuţie;
- module.

Pe această bază au fost dezvoltate interfeţe de programare (API), servicii, extensii OSGi (OSGi layers).

Cadrul de lucru conţine un model specific de aplicaţie sub formă de componentă sau modul OSGi (bundle for deployment). O asemenea componentă poate pune la dispoziţia altor componente funcţionalităţi, comportându-se ca un serviciu (ofertant de servicii) sau poate executa o acţiune punctuală. O componentă OSGi se prezintă sub forma unei arhive `jar`.

În esenţă, scopul unui cadru de lucru OSGi este oferirea unui mediu pentru crearea şi integrarea uniformă de unităţi (module, componente) de soft.

O componentă OSGi se poate instala, lansa în execuţie, opri, actualiza şi dezinstala.

Cadrul de lucru conţine un *registru de servicii* care permite unei componente OSGi să sesizeze existenţa, apariţia sau dispariţia unor servicii.

Programarea unei aplicaţii (serviciu) OSGi se poate face în mod

- *imperativ* prin existenţa unei clase *activator* ce implementează interfaţa `org.osgi.framework.BundleActivator`.

- *declarativ* prin utilizarea unor resurse OSGi suplimentare care înlocuiesc *activatorul* cu fişiere de configurare. *Descriptive service, blueprint, iPOJO* sunt tehnologii de programare declarative.

Cadre de lucru OSGi

Există mai multe implementări a modelului OSGi, dintre care amintim *apache-felix, apache-karaf, equinox* din *eclipse, Knopflerfish.* Serverul de aplicaţii *glassfish* înglobează *apache-felix.*

Exemplificarea noastră se bazează pe *apache-felix.*

OSGi prin *apache-felix*

Instalarea constă în dezarhivarea arhivei descărcate din Internet.

Utilizare. Din catalogul unde s-a instalat *apache-felix*, mediul se lansează prin

```
java -jar bin\felix.jar
```

În catalogul în care s-a instalat *apache-felix* se va crea un subcatalog *felix-cache* care este folosit de cadrul de lucru.

Comenzile OSGi sunt:

Comanda	Funcţionalitatea
`lb`	Afişează lista modulelor OSGi instalate.
`exit <int>`	Părăseşte şi închide cadrul de lucru.
`install file:`*modulOSGi*`.jar`	Instalează modulul OSGi La instalare unui modul i se atribuie în vederea identificării un număr natural *id.*
`start` *id*	Lansează modulul OSGi *id.*
`start file:` *modulOSGi*`.jar`	Instalează şi lansează modulul OSGi
`stop` *id*	Opreşte modulul OSGi *id.*
`uninstall` *id*	Dezinstalează modulul OSGi *id.*

Interfaţa de lucru, *Apache Felix Gogo,* implementează RFC (*Request for Comments*) 147 publicat de *Internet Engineering Task Force* (IETF).

Modelul de programare imperativ

Elementul cu care operează cadrul de lucru OSGi este modulul sau componenta OSGi (*bundle*). În modelul de programare imperativ un modul OSGi este compus din

1. O clasă ce implementează interfaţa `BundleActivator.`

   Interfaţa `BundleActivator` declară metodele

   - `public void start(BundleContext` *ctx*`)`
     Fixează activităţile executate la lansarea modulului OSGi.

- `public void stop(BundleContext` *ctx*`)`
  Fixează activităţile executate la oprirea modulului OSGi.

Această clasă poate lipsi în cazul declarării unei interfaţe.

2. Un fişier text *manifest*`.mf` de proprietăţi (nume: valoare) ale modulului OSGi, cu extensia `mf`. Dintre proprietăţi menţionăm:

   - `Bundle-Name:` numele componentei OSGi
   - `Bundele-Description:` descrierea componentei OSGi
   - `Bundle-Version:` 1.0.0
   - `Bundle-Activator:` clasa care implementează interfaţa `BundleActivator`
   - `Bundle-Classpath:` variabila *classpath* utilizată de componenta OSGi. Uzual, primul element, indicat prin "." (punct) va desemna catalogul curent.
   - `Import-Package:` Lista pachetelor utilizate, furnizate de alte componente OSGi.
   - `Export-Package:` Lista pachetelor cuprinse în modulul OSGi.
     Fiecărui pachet trebuie să i se atribuie o versiune, de exemplu

     ```
     Export-Package: mathlib.client.cvadra;version="1.0.0",mathlib.client.ecalg;
     version="1.0.0"
     ```

   Ultima linie din fişierul *manifest*`.mf` este o linie vidă. În liste, separatorul este virgula.

Rezolvarea problemelor prin module OSGi

Considerăm clasa *TestEcAlg*

```
1  import mathlib.client.ecalg.*;
2  import mathlib.client.ecalg.impl.*;
3  import org.osgi.framework.*;

5  public class Activator implements BundleActivator{
6    public void start(BundleContext context){
7      DataIn din=new SimpluEcAlgDataIn();
8      IMetodaTangentei obj=new MetodaTangentei();
9      DataOut dout=obj.metodaTangentei(din);
10     System.out.println("\nIndicatorul de raspuns : "+dout.getInd());
11     System.out.println("Solutia ecuatiei : "+dout.getX());
12     System.out.println("Valoarea functiei in solutie : "+dout.getF());
13     System.out.println("Numarul iteratiilor efectuate : "+dout.getNi());
14   }

16   public void stop(BundleContext context) {}
17 }
```

şi după compilare, arhivăm cu `jar cfvm testecalg.jar manifest.mf ...` ansamblul

```
|--> lib
|    |   mathlib.jar
|    Activator.class
|    SimpluEcAlgDataIn.class
```

Fişierul *manifest.mf* este

```
Bundle-Name: TestEcAlg
Bundle-Description: Test EcAlg
Bundle-Version: 1.0.0
Bundle-Activator: Activator
Bundle-Classpath: .,lib/mathlib.jar
Import-Package: org.osgi.framework
```

După lansarea cadrului OSGi în execuţie, instalăm componenta OSGi astfel creată şi o lansăm în lucru:

```
g! start file: . . .\testecalg.jar

Indicatorul de raspuns : 0
Solutia ecuatiei : -0.766664695962095
Valoarea functiei in solutie : 1.1102230246251565E-16
Numarul iteratiilor efectuate : 5
```

Pentru a relua execuţia trebuie să aflăm codul atribuit de cadrul de lucru OSGi componentei *testecalg.jar*

```
g! lb
  ID|State     |Level|Name
  . . .
   5|Active    |    1|TestEcAlg(1.0.0)|1.0.0
```

după care componenta trebuie oprită

```
g! stop 5
```

Dezinstalarea componentei de cadrul de lucru OSGi se obţine prin

```
g! uninstall 5
```

O altă variantă de programare constă în utilizarea *serviciilor OSGi.* În cele ce urmează vom pleca de la mini-biblioteca *mathlib* realizată şi arhivată în fişierul *mathlib.jar.*

Se va crea o componentă OSGi care va înregistra câte un serviciu pentru fiecare interfeţă *IMetodaTangentei* şi *IMetodaSimpson.* Totodată, componenta OSGi, corespunzătoare interfeţelor, va exporta pachetele
mathlib.client.cvadra;version="1.0.0",mathlib.client. ecalg;version="1.0.0".

Înregistrarea unui serviciu se face prin intermediul metodei
`ServiceRegistration registerService(String, Object, Properties)`
a clasei `BundleContext,` unde corespunzător variabilei de tip

- `String` se află numele interfeţei, care fixează astfel numele serviciului;
- `Object` se află o instanţă a clasei care implementează interfaţa;
- `Properties` se află elemente pentru identificarea serviciului, în cazul în care mai multe componente OSGi generează servicii aceleiaşi interfeţe.

În cazul exemplului, codul clasei *Activator.java* este

```
1  import mathlib.client.cvadra.*;
2  import mathlib.client.ecalg.*;
3  import mathlib.client.cvadra.impl.*;
4  import mathlib.client.ecalg.impl.*;
5  import org.osgi.framework.*;

7  public class Activator implements BundleActivator{
8    ServiceRegistration metodaSimpsonService;
9    ServiceRegistration metodaTangenteiService;
10   public void start(BundleContext context){
11     metodaSimpsonService=
12       context.registerService(IMetodaSimpson.class.getName(),
13         new MetodaSimpson(),null);
14     System.out.println("Registering MetodaSimpson service.");
15     metodaTangenteiService=
16       context.registerService(IMetodaTangentei.class.getName(),
17         new MetodaTangentei(),null);
18     System.out.println("Registering MetodaTangentei service.");
19   }

21   public void stop(BundleContext context) {
22     metodaSimpsonService.unregister();
23     metodaTangenteiService.unregister();
24   }
25 }
```

iar fişierul *manifest.mf* este

```
1  Bundle-Name: MathlibService
2  Bundle-Description: Mathlib Service
3  Bundle-Version: 1.0.0
4  Bundle-Activator: Activator
5  Bundle-Classpath: .,lib/mathlib.jar,lib/log4j-1.2.15.jar
6  Import-Package: org.osgi.framework
7  Export-Package: mathlib.client.cvadra;version="1.0.0",
8    mathlib.client.ecalg;version="1.0.0"
```

Desfăşurarea resurselor în vederea arhivării este

```
|--> lib
|    |   mathlib.jar
|    Activator.class
```

Rezolvarea fiecărei probleme se obţine în câte un modul OSGi. Vom reutiliza clasele *SimpleEcAlgDataIn* şi *SimpluCvadraDataIn*, la care se adaugă câte o clasă care implementează interfaţa `BundleActivator`.

Calculul soluţiei negative a ecuaţiei $2^x - x^2 = 0$

Un obiect care implementează interfaţa se obţine în doi paşi:

1. Se găseşte o referinţa a serviciului cu metoda

   `ServiceReference getServiceReference(String)`

   a clasei `BundleContext`. Variabila `String` reprezintă numele serviciului, adică numele interfeţei.

2. Se obţine o instanţă a clasei ce implementează interfaţa cu metoda

   `Object getService(ServiceReference)`

Codul clasei *Activator.java* este

```
import mathlib.client.ecalg.*;
import org.osgi.framework.*;

public class Activator implements BundleActivator{
  ServiceReference metodaSimpsonServiceReference;
  public void start(BundleContext context){
    try{
      metodaSimpsonServiceReference=
        context.getServiceReference(IMetodaTangentei.class.getName());
      if(metodaSimpsonServiceReference!=null){
        DataIn din=new SimpluEcAlgDataIn();
        IMetodaTangentei obj=
         (IMetodaTangentei)context.getService(metodaSimpsonServiceReference);
        DataOut dout=obj.metodaTangentei(din);
        System.out.println("\nIndicatorul de raspuns : "+dout.getInd());
        System.out.println("Solutia ecuatiei : "+dout.getX());
        System.out.println("Valoarea functiei in solutie : "+dout.getF());
        System.out.println("Numarul iteratiilor efectuate : "+dout.getNi());
      }
    }
    catch(Exception e){
      System.out.println("App Exception : "+e.getMessage());
    }
  }

  public void stop(BundleContext context){
    if(metodaSimpsonServiceReference!=null){
      context.ungetService(metodaSimpsonServiceReference);
    }
  }
}
```

Fişierul *manifest.mf* corespunzător este

```
Bundle-Name: AppEcAlg
Bundle-Description: Rezolvarea unei ecuatii algebrice
Bundle-Version: 1.0.0
Bundle-Activator: Activator
Import-Package: org.osgi.framework,mathlib.client.ecalg
```

Dacă *service.jar* şi *appecalg.jar* sunt arhivele corespunzătoare componentelor OSGi *MathlibService* şi respectiv *AppEcAlg* (după valoarea atributului `Bundle-Name`) atunci comenzile OSGi pentru rularea aplicaţiei sunt

```
g! start file: . . .\service.jar
  Registering MetodaSimpson service.
  Registering MetodaTangentei service.
g! start file: . . .\appecalg.jar
```

Calculul integralei $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x$

În mod asemănător, dar folosind altă modalitate de programare OSGi, utilizăm

Clasa *Activator.java*

```
import mathlib.client.cvadra.*;
import org.osgi.framework.*;
import org.osgi.util.tracker.ServiceTracker;

public class Activator implements BundleActivator{
  ServiceTracker metodaSimpsonServiceTracker;
  public void start(BundleContext context){
```

```
8       metodaSimpsonServiceTracker=new ServiceTracker(context,
9           IMetodaSimpson.class.getName(),null);
10      metodaSimpsonServiceTracker.open();
11      DataIn din=new SimpluCvadraDataIn();
12      IMetodaSimpson obj=
13        (IMetodaSimpson)metodaSimpsonServiceTracker.getService();
14      DataOut dout=obj.metodaSimpson(din);
15      System.out.println("\nIndicatorul de raspuns : "+dout.getInd());
16      System.out.println("Integrala : "+dout.getIntegrala());
17      System.out.println("Numarul iteratiilor efectuate : "+dout.getNi());
18    }

20    public void stop(BundleContext context){
21      metodaSimpsonServiceTracker.close();
22    }
23  }
```

împreună cu *manifest.mf*

```
1 Bundle-Name: AppIntegrala
2 Bundle-Description: Calculul unei integrale
3 Bundle-Version: 1.0.0
4 Bundle-Activator: Activator
5 Import-Package: org.osgi.framework,org.osgi.util.tracker,
6   mathlib.client.cvadra
```

Capitolul 2

Accesarea în Java a unor produse matematice

Multe pachete de programe matematice oferă posibilitatea apelării lor din clase Java. Ne vom limita numai la produse distribuite gratuit. Vom prezenta utilizarea în Java a produselor *Mathematica, Maple, Scilab.*

Ca motivaţii pentru un asemenea interes este posibilitatea elaborării unei aplicaţii Java care utilizează funcţionalităţi ale softurilor amintite.

2.1 Java cu *Mathematica*

Mathematica este unul din produsele de vârf de matematică cu facilităţi de calcul simbolic şi numeric, de grafică şi de dezvoltare - programare proprie. *Mathematica* este un produs comercial realizat de *Wolfram Research.*

Legătura dintre *Mathematica* şi Java este asigurată de componenta `JLink` a produsului. `JLink` permite utilizarea unei clase Java într-o sesiune *Mathematica* şi a resurselor *Mathematica* într-o clasă Java. Această din urmă posibilitate va fi prezentată în continuare.

Compilarea ca şi execuţia presupune declararea în variabila `classpath` a fişierului *JLink.jar* din distribuţia *Mathematica.*

Într-o clasă Java, `JLink` se declară prin

```
import com.wolfram.jlink.*;
```

Accesarea resurselor *Mathematica* se face prin intermediul unui obiect de tip `KernelLink`, a cărei instanţiere poate fi

```
KernelLink ml=null;
try{
  String[] mlArgs = {"-linkmode", "launch", "-linkname", args[0]};
  ml=MathLinkFactory.createKernelLink(mlArgs);
  ml.discardAnswer();
}
```

```
catch(MathLinkException e){
  System.out.println("Fatal opening link error : "+e.getMessage());
  System.exit(1);
}
```

Ultimul element al şirului *mlArgs* fixează locaţia nucleului *MathKernel.exe,* fiind transmis ca argument al programului Java.

Şablonul de prelucrare al unei expresii / comenzi *Mathematica* este

```
try{
  // Evaluarea expresiei / comenzii reprezentata prin String-ul expr
  String expr=". . .";
  ml.evaluate(expr);
  ml.waitForAnswer();
  // Prelucrarea rezultatului
 }
catch(MathLinkException e){
  System.out.println("MathLinkException : "+e.getMessage());
}
finally{
  ml.close();
}
```

Alternativ, metoda `evaluate` poate avea ca parametru o variabilă de tip `Expr`.

Funcţie de tipul rezultatului, acesta se obţine cu una din metodele clasei `KernerLink`: `Expr getExpr(), double getDouble(), int getInteger(), boolean getBoolean()`.

Exemplul 2.1.1 *Să se rezolve ecuaţia* $2^x - x^2 = 0$.

```
1  import com.wolfram.jlink.*;
2  public class NSolve{
3    public static void main(String[] args){
4      KernelLink ml=null;
5      try{
6        String[] mlArgs = {"-linkmode", "launch", "-linkname", args[0]};
7        ml = MathLinkFactory.createKernelLink(mlArgs);
8      }
9      catch(MathLinkException e){
10       System.out.println("Fatal opening link error : "+e.getMessage());
11       System.exit(1);
12     }
13     try{
14       ml.discardAnswer(); //f. important
15       ml.evaluate("NSolve[2^x-x^2==0,x]");
16       ml.waitForAnswer();
17       Expr result=ml.getExpr();
18       System.out.println(result.toString());
19     }
20     catch(Exception e){
21       System.out.println("MathLinkException : "+e.getMessage());
22     }
23     finally{
24       ml.close();
25     }
26   }
27 }
```

Execuţia se comandă prin

```
java NSolve "d:/Wolfram Research/Mathematica/5.1/MathKernel.exe"
```

iar rezultatul obţinut este

```
{{Rule[x,-0.766664695962123]},{Rule[x,2.0]},{Rule[x,4.0]}}
```

Extragerea soluţiilor se poate programa utilizând metodele clasei `Expr`, prin

```
for(int i=1;i<=result.length();i++){
  Expr e1=result.part(i);
  Expr e2=e1.part(1);
  Expr[] e3=e2.args();
  System.out.println(e3[1].toString());
}
```

Vizualizarea imaginilor grafice produse de *Mathematica* se face prin intermediul obiectelor care instanţiază clasele:

- `MathGraphicsJPanel`,

  ce extinde clasa `javax.swing.JFrame`;

- `MathCanvas`

  ce extinde clasa `java.awt.Canvas`.

Ambele clase conţin metodele:

- `void setImageType(int` *type*`)`

  Fixează natura răspunsului `GRAPHICS` sau `TYPESET`.

- `void setUsesFE(boolean` *useFE*`)`

  Fixează modul de realizare a imaginii pe ecran, cu *Mathematica* sau nu.

- `void setMathCommand(String` *cmd*`)`

  Fixează funcţia *Mathematica* care se evaluează şi a cărui rezultat se afişează.

- `Image getImage()`

  Returnează imaginea grafică construită într-o variabilă de tip `Image`.

Exemplul 2.1.2 *Program Java care afişează reprezentările grafice realizate de Mathematica.*

Utilizând pachetul `javax.swing`, codul sursă este

```
1 import com.wolfram.jlink.*;

3 public class MathGraphics extends MathJFrame{
4   private MathGraphicsJPanel mathGraphicsJPanel;
5   private javax.swing.JLabel mJLabel;
6   private javax.swing.JTextArea mJTextArea;
7   private javax.swing.JButton mJButton;
8   private javax.swing.JScrollPane mJScrollPane;
```

```
10    static KernelLink ml;

12    public MathGraphics(){
13      initComponents();
14    }

16    private void initComponents(){
17      java.awt.GridBagConstraints gridBagConstraints;
18      setTitle("Mathematica : Grafica");
19      setSize(400,400);
20      getContentPane().setLayout(new java.awt.GridBagLayout());

22      mathGraphicsJPanel=new MathGraphicsJPanel(ml);
23      mathGraphicsJPanel.setBackground(java.awt.Color.white);
24      mathGraphicsJPanel.setPreferredSize(new java.awt.Dimension(250,200));
25      gridBagConstraints = new java.awt.GridBagConstraints();
26      gridBagConstraints.gridx = 1;
27      gridBagConstraints.gridy = 0;
28      getContentPane().add(mathGraphicsJPanel, gridBagConstraints);

30      mJLabel=new javax.swing.JLabel();
31      mJLabel.setPreferredSize(new java.awt.Dimension(200,18));
32      mJLabel.setText("Expresie \"Mathematica\"");
33      mJLabel.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
34      gridBagConstraints = new java.awt.GridBagConstraints();
35      gridBagConstraints.gridx = 0;
36      gridBagConstraints.gridy = 0;
37      getContentPane().add(mJLabel, gridBagConstraints);

39      mJScrollPane=new javax.swing.JScrollPane();
40      mJScrollPane.setPreferredSize(new java.awt.Dimension(200,60));
41      mJTextArea=new javax.swing.JTextArea();
42      mJScrollPane.setViewportView(mJTextArea);
43      gridBagConstraints = new java.awt.GridBagConstraints();
44      gridBagConstraints.gridx = 0;
45      gridBagConstraints.gridy = 1;
46      getContentPane().add(mJScrollPane, gridBagConstraints);

48      mJButton=new javax.swing.JButton();
49      mJButton.setText("Evalueaza");
50      mJButton.setPreferredSize(new java.awt.Dimension(120,18));
51      mJButton.addMouseListener(new java.awt.event.MouseAdapter() {
52        public void mouseClicked(java.awt.event.MouseEvent evt) {
53          mJButtonMouseClicked(evt);
54        }
55      });
56      gridBagConstraints = new java.awt.GridBagConstraints();
57      gridBagConstraints.gridx = 0;
58      gridBagConstraints.gridy = 2;
59      getContentPane().add(mJButton, gridBagConstraints);

61      addWindowListener(new java.awt.event.WindowAdapter() {
62        public void windowClosing(java.awt.event.WindowEvent evt) {
63          if(ml!=null){
64            ml.evaluateToInputForm("CloseFrontEnd[]", 0);
65            ml.close();
66          }
67          dispose();
68          System.exit(0);
69        }
70      });
71      ml.evaluateToInputForm("Needs[\""+KernelLink.PACKAGE_CONTEXT+"\"]",0);
72      ml.evaluateToInputForm("ConnectToFrontEnd[]", 0);
73      toFront();
74      pack();
75    }
```

```
77    private void mJButtonMouseClicked(java.awt.event.MouseEvent evt){
78      mathGraphicsJPanel.setImageType(MathGraphicsJPanel.GRAPHICS);
79      mathGraphicsJPanel.setUsesFE(true);
80      mathGraphicsJPanel.setMathCommand(mJTextArea.getText());
81    }

83    public static void main(String args[]) {
84      try {
85        String[] mlArgs = {"-linkmode", "launch", "-linkname", args[0]};
86        ml = MathLinkFactory.createKernelLink(mlArgs);
87        ml.discardAnswer();
88      }
89      catch (MathLinkException e) {
90        System.out.println("An error occurred connecting to the kernel.");
91        if (ml != null)
92          ml.close();
93        return;
94      }
95      java.awt.EventQueue.invokeLater(new Runnable() {
96        public void run() {
97          new MathGraphics().setVisible(true);
98        }
99      });
100   }
101 }
```

Clientul interacţionează cu programul prin interfaţa grafică din Fig. 2.1. În zona *TextArea* se introduce o comandă *Mathematica.*

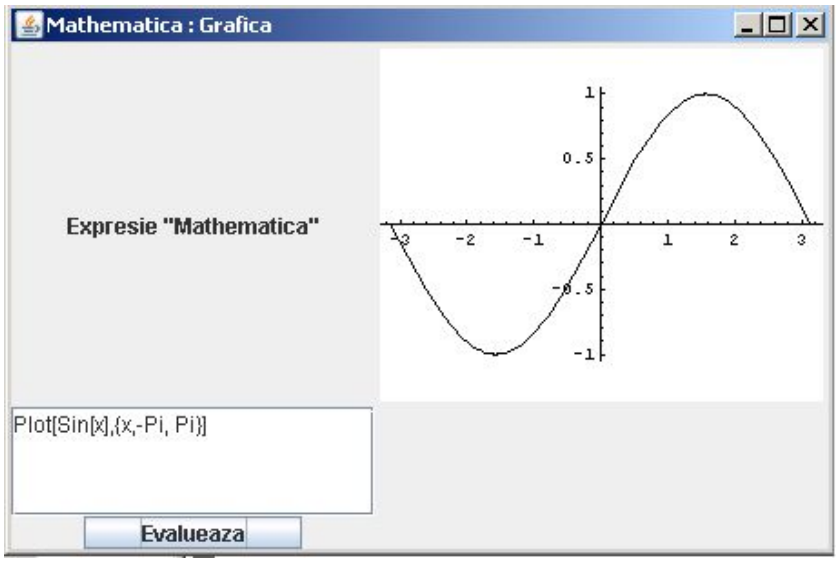


Fig. 2.1: Reprezentare grafică produsă de *Mathematica.*

2.2 Java cu *Maple*

Maple este un produs comercial de matematică produs de *Waterloo Maple Inc.,* aflat în concurenţă cu *Mathematica.*

Legătura dintre Java şi *Maple* se stabileşte prin interfaţa de programare (Application Program Interface – API) *openmaple.* Problema de care ne ocupăm constă în utilizarea facilităţilor oferite de *Maple* în programe Java.

Utilizarea lui *openmaple* într-un program Java se declară prin

```
import com.maplesoft.openmaple.*;
import com.maplesoft.externalcall.MapleException;
```

Principala clasă prin care un program Java interacţionează cu *Maple* este

`com.maplesoft.openmaple.Engine`

Constructorul clasei este

`Engine(String[]` *args,* `EngineCallBack` *cb,* `Object` *userData,* `Object` *res*`)`

unde

- *args*={*"java"*};
- *cb* este o instanţă a unei clase ce implementează interfaţa `EngineCallBack.` Interfaţa declară metode care fixează răspunsurile la apelurile emise de nucleul *Maple* (apeluri inverse);

  openmath oferă clasa `EngineCallBackDefault` ca o implementare uzuală a interfeţei `EngineCallBack.` Orice mesaj / rezultat furnizat de *Maple* este afişat;
- *userData* este un parametru care conţine date ce se includ în *cb* la fiecare apel invers;
- *res=null* - este un argument neutilizat.

Dintre metodele clasei `Engine` amintim:

- `Algebraic evaluate(String` *comandaMaple*`)`

  Clasa `Algebraic` este o clasă acoperitoare în Java a obiectelor *Maple.*

Dintre metodele clasei `Algebraic` semnalăm:

- `String toString()`

  Converteşte obiectul *Maple* în string. Dacă se aplică acestui string metoda `evaluate` atunci se reobţine obiectul *Maple.*

Reluăm exerciţiul

Exemplul 2.2.1 *Să se rezolve ecuaţia* $2^x - x^2 = 0$.

Codul programului este

```
1  import com.maplesoft.openmaple.*;
2  import com.maplesoft.externalcall.MapleException;
3  class Solve{
4    public static void main( String args[] ){
5      try{
6        String[] argsMaple={"java"};
7        Engine engine=new Engine(argsMaple,new EngineCallBacksDefault(),
8          null,null);
9        Algebraic sol=engine.evaluate("u:=[evalf(solve(2^x-x^2,x))];");
10       Algebraic an=engine.evaluate("nops(u);");
11       int n=(new Integer(an.toString())).intValue();
12       double[] x=new double[n];
13       for(int i=1;i<=n;i++){
14         Algebraic ax=engine.evaluate("u["+i+"];");
15         x[i-1]=(new Double(ax.toString())).doubleValue();
16       }
17       System.out.println("Solutiile obtinute");
18       for(int i=0;i<n;i++)
19         System.out.println("x["+(i+1)+"] = "+x[i]);
20     }
21     catch (MapleException e){
22       System.out.println("MapleException : "+e.getMessage());
23       return;
24     }
25   }
26 }
```

cu rezultatele

```
u := [2., 4., -.7666646958]
3
2.
4.
-.7666646958
Solutiile obtinute
x[1] = 2.0
x[2] = 4.0
x[3] = -0.7666646958
```

Primele cinci linii sunt afişate de obiectul anonim de tip `EngineCallBackDefault`.

Soluţa negativă se putea obţine direct prin

```
engine.evaluate("solve(2^x-x^2,x,x=-infinity..0);");
```

Semnalăm faptul că pentru compilare şi execuţie trebuie să includem în variabila de sistem `classpath` fişierele *jopenmaple.jar, externalcall.jar* din catalogul `Maple_HOME\java`.

În plus, pentru execuţie, trebuie completată variabila de sistem `PATH` cu calea către catalogul `Maple_HOME\bin.X86_64_WINDOWS`.

2.3 Java cu *Scilab*

Scilab este un produs de calcul numeric produs de INRIA (*Institut National de Recherche en Informatique et Automatique*) – Franţa – disponibil în mediile Windows, Linux şi Mac OS X. Produsul este distribuit gratuit. *Scilab* este însoţit de o documentaţie cuprinzătoare.

Din punctul de vedere al utilizări *Scilab* are trăsături comune cu produsul comercial *Matlab.*

Prezentarea se va rezuma la versiunea a doua a puntii oferită de *Scilab* către Java.

Funcţiile *Scilab* pot fi apelate dintr-un program Java, conexiunea cu Java se bazează pe pachetele

No.	Resursa jar	Locaţia
1	`org.scilab.modules.javasci.jar`	SCILAB_HOME\modules\javasci\jar
2	`org.scilab.modules.types.jar`	SCILAB_HOME\modules\types\jar

Compilarea şi **execuţia** necesită prezenţa în variabila sistem `classpath` a referinţei către fişerele `jar` menţionate mai sus.

Vom utiliza unele din clasele şi interfeţele:

```
org.scilab.modules.javasci.Scilab
org.scilab.modules.javasci.JavasciException
org.scilab.modules.types.ScilabType
org.scilab.modules.types.ScilabBoolean
org.scilab.modules.types.ScilabDouble
org.scilab.modules.types.ScilabInteger
org.scilab.modules.types.ScilabString
org.scilab.modules.types.ScilabList
```

Prezentăm succint elementele care intervin în exemplele pe care le vom da:

Clasa `org.scilab.modules.javasci.Scilab`

Constructor:

- `public Scilab()`

Clasa `Scilab` conţine metodele statice:

- `public boolean open() throws JavasciException`
- `public boolean close()`
- `public boolean exec(String` *sarcină*`)`
- `public boolean exec(String[]` *sarcini*`)`
- `public boolean exec(File` *numeFişierScript*`)`
- `public boolean put(String` *numeVar*`, ScilabType` *val*`)`

  Se defineşte variabila *Scilab* *numeVar* a cărei valoare este *val.*
- `ScilabType get(String` *numeVar*`)`

  Returnează un obiect acoperitor Java corespunzător tipului variabilei *Scilab.*

Valoarea logică returnată de metodele de mai sus este `true` dacă *Scilab* nu generează o eroare.

Şablonul de apelare pentru execuţia de cod *Scilab* este

```
import org.scilab.modules.javasci.Scilab;

public class ApelScilab{
  public static void main(String[] args){
    try{
      Scilab sci=new Scilab();
      if(sci.open()){
        . . .
        sci.exec( cod Scilab );
        . . .
        sci.close();
      }
      else{
        System.out.println("Could not start Scilab ");
      }
    }
    catch(org.scilab.modules.javasci.JavasciException e) {
       System.err.println("Exception: " + e.getLocalizedMessage());
    }
  }
}
```

Interfaţa `org.scilab.modules.types.ScilabType`

Interfaţa declară metodele:

- `boolean equals(Object` *obiect*`)`
- `int getHeight()`

  Returnează numărul liniilor.
- `int getWidth()`

  Returnează numărul coloanelor.
- `boolean isEmpty()`
- `String toString()`

Interfaţa este implementată de clasele `ScilabBoolean, ScilabDouble, ScilabInteger, ScilabString, ScilabList.`

Clasa `org.scilab.modules.types.ScilabDouble`

Constructori:

- `ScilabDouble()`

- `ScilabDouble(double` *data*`)`
- `ScilabDouble(double[][]` *data*`)`
- `ScilabDouble(double` *realData,* `double` *imagData*`)`
- `ScilabDouble(double[][]` *realData,* `double[][]` *imagData*`)`

Metode:

- `double[][] getRealPart()`
- `double[][] getImaginarylPart()`
- `boolean isReal()`

Schema pentru transferul unei date `double` x din Java în *Scilab* constă din

```
double x=. . .;
ScilabDouble sci_x=new new ScilabDouble(x);
sci.put("x",sci_x);
```

Invers, preluarea în Java a valorii unei variabile *Scilab a,* presupusă număr real, se programează prin

```
ScilabDouble sci_a=(ScilabDouble)sci.get("a");
double[][] aa=sci_a.getRealPart();
double a=aa[0][0];
```

Exemplul 2.3.1 *Execuţia unui script Scilab u=1,v=u+1, disp(v).*

```
1  import org.scilab.modules.javasci.Scilab;
2  import org.scilab.modules.types.ScilabType;
3  import org.scilab.modules.types.ScilabDouble;

5  public class  TstJavaSci1{
6    public static void main(String[] args){
7      try{
8        Scilab sci=new Scilab();
9        if(sci.open()){

11         sci.exec(new String[]{"u=1","v=u+1","disp(v)"});

13         sci.close();
14       }
15       else{
16         System.out.println("Could not start Scilab ");
17       }
18     }
19     catch(org.scilab.modules.javasci.JavasciException e) {
20       System.err.println("An exception occured: " + e.getLocalizedMessage());
21     }
22   }
23 }
```

Se obţine

2.

Exemplul 2.3.2 *Calculul soluţiei negative a ecuaţiei* $2^x - x^2 = 0$.

```
import org.scilab.modules.javasci.Scilab;
import org.scilab.modules.types.ScilabType;
import org.scilab.modules.types.ScilabDouble;

public class  TstJavaSci2{
  public static void main(String[] args){
    try{
      Scilab sci=new Scilab();
      if(sci.open()){
        String[] s={"deff(\'y=fct(x)\',\'y=2.^x-x.*x\')","[x,y,info]=fsolve(-0.5,fct)"};
        sci.exec(s);
        ScilabDouble sci_x=(ScilabDouble)sci.get("x");
        double[][] x=sci_x.getRealPart();
        System.out.println("Solutia : "+x[0][0]);

        ScilabDouble sci_y=(ScilabDouble)sci.get("y");
        double[][] y=sci_y.getRealPart();
        System.out.println("Valoarea functiei in solutie : "+y[0][0]);

        ScilabDouble sci_info=(ScilabDouble)sci.get("info");
        double[][] info=sci_info.getRealPart();
        System.out.println("Indicatorul de raspuns : "+info[0][0]);
        sci.close();
      }
      else{
        System.out.println("Could not start Scilab ");
      }
    }
    catch(org.scilab.modules.javasci.JavasciException e) {
      System.err.println("An exception occured: " + e.getLocalizedMessage());
    }
  }
}
```

Rezultatele obţinute sunt

```
Solutia : -0.766664695962123
Valoarea functiei in solutie : 1.1102230246251565E-16
Indicatorul de raspuns : 1.0
```

Exemplul 2.3.3 *Calculul integralei* $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x$.

```
import org.scilab.modules.javasci.Scilab;
import org.scilab.modules.types.ScilabType;
import org.scilab.modules.types.ScilabDouble;

public class  TstJavaSci3{
  public static void main(String[] args){
    try{
      Scilab sci=new Scilab();
      if(sci.open()){
        String fct="\'log(1+tan(x))\'";
        String var="\'x\'";
        String lowerBound="0";
        String upperBound="%pi/4";
        String scicomm="u=integrate("+fct+","+var+","+lowerBound+","+upperBound+")";
        boolean error=sci.exec(scicomm);
        System.out.println("Operatia s-a terminat cu succes (true/false) : "+error);
        ScilabDouble sci_u=(ScilabDouble)sci.get("u");
```

```
18          double[][] u=sci_u.getRealPart();
19          System.out.println("Integrala : "+u[0][0]);
20          sci.close();
21        }
22        else{
23          System.out.println("Could not start Scilab ");
24        }
25      }
26      catch(org.scilab.modules.javasci.JavasciException e) {
27        System.err.println("An exception occured: " + e.getLocalizedMessage());
28      }
29    }
30  }
```

Rezultatele sunt:

```
Operatia s-a terminat cu succes (true/false) : true
Integrala : 0.27219826128795027
```

Exemplul 2.3.4 *Utilizarea numerelor complexe. Pentru* $x = 1+2i, y = 1+i$ *se calculează produsul* $x \cdot y$.

```
1  import org.scilab.modules.javasci.Scilab;
2  import org.scilab.modules.types.ScilabType;
3  import org.scilab.modules.types.ScilabDouble;

5  public class  TstJavaSci4{
6    public static void main(String[] args)  {
7      try{
8        Scilab sci=new Scilab();
9        if(sci.open()){
10         double[][] re={{1},{1}};
11         double[][] im={{2,},{-1}};
12         ScilabDouble sci_z=new ScilabDouble(re,im);
13         sci.put("z",sci_z);
14         sci.exec("w=z(1)*z(2)");
15         ScilabDouble sci_w=(ScilabDouble)sci.get("w");
16         double[][] w_re=sci_w.getRealPart();
17         double[][] w_im=sci_w.getImaginaryPart();
18         System.out.println(w_re[0][0]+" I* "+w_im[0][0]);

20         sci.close();
21       }
22       else{
23         System.out.println("Could not start Scilab ");
24       }
25     }
26     catch(org.scilab.modules.javasci.JavasciException e) {
27       System.err.println("An exception occured: " + e.getLocalizedMessage());
28     }
29   }
30 }
```

Exemplul 2.3.5 *Rezolvarea sistemul algebric de ecuaţii liniare*

$$\begin{cases} x_1 + x_2 + x_3 + x_4 = 2 \\ 2x_1 - x_2 + 2x_3 - x_4 = 1 \\ x_1 + 2x_2 - x_3 + 2x_4 = -1 \\ 2x_1 + x_2 + 4x_3 + x_4 = 7 \\ 3x_1 + 2x_2 - 2x_3 + 2x_4 = -5 \end{cases}$$

având soluţia $x_1 = -1, x_2 = 1 - x_4, x_3 = 2$.

Sub formă matriceală soluţia se scrie

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ -1 \\ 0 \\ 1 \end{pmatrix} c = \begin{pmatrix} -1 \\ 1 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \frac{-1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} c'. \tag{2.1}$$

unde $c \in \mathbb{R}$.

În *Scilab,* rezolvarea unui sistem algebric de ecuaţii liniare $Ax+b = 0, A \in M_{m,n}(\mathbb{R}), b \in \mathbb{R}^n$, se obţine cu funcţia `[x,k]=linsolve(A,b)`. x este o soluţie particulară a sistemului în timp ce k este o matrice având drept coloane o bază ortonormată a nucleului $\mathrm{Ker}(A) = \{x \in \mathbb{R}^n : Ax = 0\}$. Dacă sistemul este incompatibil atunci rezultatele sunt $x = [\,], k = [\,]$.

```
import org.scilab.modules.javasci.Scilab;
import org.scilab.modules.types.ScilabType;
import org.scilab.modules.types.ScilabDouble;

public class  TstJavaSci5{
  public static void main(String[] args)  {
    try{
      Scilab sci=new Scilab();
      if(sci.open()){
        double[][] a={{1,1,1,1},{2,-1,2,-1},{1,2,-1,2},{2,1,4,1},{3,2,-2,2}};
        ScilabDouble sci_a=new ScilabDouble(a);
        sci.put("a",sci_a);
        double[][] b={{2},{1},{-1},{7},{-5}};
        ScilabDouble sci_b=new ScilabDouble(b);
        sci.put("b",sci_b);
        //boolean error=sci.exec("[x,k]=linsolve("+a+",-"+b+")");
        boolean error=sci.exec("[x,k]=linsolve(a,-b)");
        System.out.println("Operatia s-a terminat cu succes (true/false) : "+error);
        ScilabDouble sci_x=(ScilabDouble)sci.get("x");
        if(!sci_x.isEmpty()){
          double[][] x=sci_x.getRealPart();
          System.out.println("x:");
          for(int i=0;i<x.length;i++){
             for(int j=0;j<x[0].length;j++)
                System.out.print(x[i][j]+" ");
             System.out.println();
          }
        }
        ScilabDouble sci_k=(ScilabDouble)sci.get("k");
        if(!sci_k.isEmpty()){
          double[][] k=sci_k.getRealPart();
          System.out.println("k:");
          for(int i=0;i<k.length;i++){
             for(int j=0;j<k[0].length;j++)
                System.out.print(k[i][j]+" ");
             System.out.println();
          }
        }
        sci.close();
      }
      else{
        System.out.println("Could not start Scilab ");
      }
    }
    catch(org.scilab.modules.javasci.JavasciException e) {
      System.err.println("An exception occured: " + e.getLocalizedMessage());
    }
  }
```

```
49 }
```

Rezultatele sunt:

```
Operatia s-a terminat cu succes (true/false) : true
x:
  - 1.0000000000000002
   0.5000000000000001
   2.0000000000000013
   0.4999999999999983
k:
  8.326672684688674E-17
  - 0.7071067811865475
  - 3.0531133177191805E-16
   0.7071067811865474
```

k aproximează versorul $(0,\ \frac{1}{\sqrt{2}},\ 0,\ \frac{-1}{\sqrt{2}})^T$. Notând cu u şi v cele două soluţii particulare puse în evidenţă la scrierea soluţiei în (2.1) şi respectiv cea dată de funcţia `linsolve`, $u = (-1,\ 1,\ 2,\ 0)^T$, $\quad v = (-1,\ 0.5,\ 2,\ 0.5)^T$, vom avea

$$u - v = (0,\ 0.5,\ 0,\ -0.5)^T = \frac{1}{\sqrt{2}}k \in \mathrm{Ker}A.$$

Pentru rezolvarea unui sistem algebric de ecuaţii liniare de forma $Ax = b$, extindem mini-biblioteca *mathlib* cu o implementare bazată pe funcţia *Scilab* `linsolve`.

Extinderea este ilustrată în schema de mai jos

```
src
|--> mathlib
|    |--> client
|    |    |    . . .
|    |    |--> linear
|    |    |    |--> impl
|    |    |    |    |   RezolvitorScilab.java // implementeaza interfata
|    |    |    |    IRezolvitorScilab.java    // interfata
|    |    |    |    DataIn.java
|    |    |    |    DataOut.java
|    |    |    |    . . .
```

Codurile claselor adăugate sunt

1. Interfaţa *mathlib.client.linear.IRezolvitorScilab.java*

```
1  package mathlib.client.linear;
2  /**
3   * Interfata rezolvarii unui sistem algebric de ecuatii liniare
4   * via Scilab.
5   */
6  public interface IRezolvitorScilab{
7    /**
8     * Rezolvarea unui sistem algebric de ecuatii liniare
9     * via Scilab.
10    */
11   public DataOut rezolvitorScilab(DataIn din);
12 }
```

2. Clasa *mathlib.client.linear.DataIn.java*

```
1  package mathlib.client.linear;
2  import java.util.Vector;
3  import java.io.BufferedReader;
4  /**
5   *  Clasa acoperitoare a datelor necesare rezolvarii
6   *  unui sistem algebric de ecuatii liniare
7   */
8  public class DataIn{
9    private double[][] matrix=null;   // matricea extinsa a sistemului

11   /**
12    * Fixeaza matricea extinsa a sistemului algebric.
13    * @param matrix matricea extinsa a sistemului.
14    */
15   public void setMatrix(double[][] matrix){
16     this.matrix=matrix;
17   }
18   /**
19    * Fixeaza matricea extinsa a sistemului algebric.
20    * @param br flux care furnizeaza matricea extinsa a sistemului;
21    * datele sunt transmise pe linii.
22    */
23   public void setMatrix(BufferedReader br) throws Exception{
24     Vector<Double> v=new Vector<Double>(10);
25     try{
26       String line;
27       int m=0,n,mn;
28       do{
29         line=br.readLine();
30         if(line!=null){
31           m++;
32           String[] result=line.split(" ");
33           n=result.length;
34           for(String s :result)
35             v.addElement(new Double(s));
36         }
37       }
38       while(line!=null);
39       if(v.size()>0){
40         mn=v.size();
41         n=mn/m;
42         matrix=new double[m][n];
43         for(int i=0;i<m;i++){
44           for(int j=0;j<n;j++){
45             matrix[i][j]=((Double)v.elementAt(i*n+j)).doubleValue();
46             System.out.print(matrix[i][j]+" ");
47           }
48           System.out.println();
49         }
50       }
51     }
52     catch(Exception e){
53       throw new Exception(e.getMessage());
54     }
55   }
56   /**
57    *  Returneaza matricea sistemului.
58    */
59   public double[][] getMatrix(){
60     return matrix;
61   }
62 }
```

Matricea *matrix* conţine ansamblul $[A\ b]$.

Metoda *setMatrix(BufferedReader)* preia datele dintr-un flux. Elementele matricei extinse $[A\ b]$ se transmit în flux, pe linii. Pe o linie separatorul este spaţiul. Din flux se preiau succesiv liniile şi se contorizează numărul lor. Toate numerele se reţin într-o colecţie de date de tip `java.util.Vector`. Împărţind numărul elementelor din colecţie la numărul liniilor se găseşte numărul coloanelor. În final, se recompune matricea cu elementele din colecţie.

3. Clasa *mathlib.client.linear.DataOut.java*

```
1  package mathlib.client.linear;
2  /**
3   *  Clasa acoperitoare a rezultatelor obtinute
4   *  la rezolvarea unui sistem algebric de ecuatii liniare
5   */
6  public class DataOut {
7    private double[] x=null;    // solutie particulara
8    private double[][] k=null; // baza a spatiului liniar al
9                               // solutiilor sistemului omogen
10   private boolean compatibil;  // natura sistemului

12   /**
13    *  Returneaza o solutie particulara a sistemului.
14    */
15   public double[] getX(){
16     return x;
17   }
18   /**
19    *  Fixeaza o solutie particulara a sistemului.
20    */
21   public void setX(double[] x){
22     this.x=x;
23   }

25   /**
26    *  Returneaza o baza normalizata a solutiilor
27    *  sistemului omogen.
28    */
29   public double[][] getK(){
30     return k;
31   }
32   /**
33    *  Fixeaza o baza normalizata a solutiilor
34    *  sistemului omogen.
35    */
36   public void setK(double[][] k){
37     this.k=k;
38   }

40   /**
41    *  Returneaza natura sistemului.
42    */
43   public boolean isCompatibil(){
44     return compatibil;
45   }
46   /**
47    *  Fixeaza natura sistemului.
48    */
49   public void setCompatibil(boolean compatibil){
50     this.compatibil=compatibil;
51   }
52 }
```

4. Clasa *mathlib.client.linear.impl.RezolvitorScilab.java*

```
1  package mathlib.client.linear.impl;
2  import org.scilab.modules.javasci.Scilab;
3  import org.scilab.modules.types.ScilabType;
4  import org.scilab.modules.types.ScilabDouble;
5  import mathlib.client.linear.*;
6  /**
7   * Implementarea rezolvarii unui sistem algebric de ecuatii
8   * liniare prin apelarea functiei linsolve din Scilab
9   */
10 public class RezolvitorScilab implements IRezolvitorScilab{
11   /**
12    * Metoda de rezolvare a sistemului algebric de ecuatii liniare.
13    */
14   public DataOut rezolvitorScilab(DataIn din){
15     DataOut dout=new DataOut();
16     double[][] matrix=din.getMatrix();
17     int l=matrix.length;
18     int c=matrix[0].length;
19     double[][] a=new double[l][c-1];
20     double[][] b=new double[l][1];
21     for(int i=0;i<l;i++){
22       for(int j=0;j<c-1;j++)
23         a[i][j]=matrix[i][j];
24       b[i][0]=matrix[i][c-1];
25     }
26     try{
27       Scilab sci=new Scilab();
28       if(sci.open()){
29         ScilabDouble sci_a=new ScilabDouble(a);
30         sci.put("a",sci_a);
31         ScilabDouble sci_b=new ScilabDouble(b);
32         sci.put("b",sci_b);
33         boolean error=sci.exec("[x,k]=linsolve(a,-b)");
34         System.out.println("Success flag (true/false) : "+error);
35         ScilabDouble sci_x=(ScilabDouble)sci.get("x");
36         if(!sci_x.isEmpty()){
37           double[][] x=sci_x.getRealPart();
38           double[] x1=new double[x.length];
39           for(int i=0;i<x.length;i++)
40             x1[i]=x[i][0];
41           dout.setX(x1);
42         }
43         ScilabDouble sci_k=(ScilabDouble)sci.get("k");
44         if(!sci_k.isEmpty()){
45           double[][] k=sci_k.getRealPart();
46           dout.setK(k);
47         }
48         if(sci_x.getRealPart().length==0)
49           dout.setCompatibil(false);
50         else
51           dout.setCompatibil(true);
52         sci.close();
53       }
54       else{
55         System.out.println("Could not start Scilab ");
56       }
57     }
58     catch(org.scilab.modules.javasci.JavasciException e) {
59       System.err.println("An exception occured: " + e.getLocalizedMessage());
60     }
61     return dout;
62   }
63 }
```

Capitolul 3

Pachete Java de calcul numeric

În prezent sunt disponibile mai multe pachete de clase Java cu facilităţi de calcul numeric. Dintre cele accesibile gratuit din internet, vom prezenta doar două produse prin exemple simple.

Pentru utilizare este nevoie doar de arhiva `jar,` care se găseşte în catalogul care se obţine în urma dezarhivării fişierului descărcat din internet.

De cele mai multe ori, semnificaţia parametrilor formali ai metodelor este imediată, în plus, prezentarea lor se poate citi din descrierea claselor (apidocs).

3.1 *apache commons-math*

Dezvoltat de fundaţia apache, *commons-math* oferă o colecţie bogată de clase pentru rezolvarea problemelor uzuale de calcul numeric şi statistică. Utilizarea claselor se face potrivit unui şablon unitar, pe care-l vom pune în evidenţă prin exemplele ce urmează.

commons-math permite lucrul cu fracţii ordinare (clasa `Fraction`), numere complexe (clasa `Complex`), matrice reale (clasa `RealMatrix`).

Dintre facilităţile de calcul numeric amintim:

- Rezolvarea unei ecuaţii algebrice. Se poate alege dintre metoda bisecţiei, metoda secantei, metoda tangentei (Newton).

  Exemplul 3.1.1 *Utilizând metoda tangentei (1.1), să se calculeze soluţia negativă a ecuaţiei* $2^x - x^2 = 0$.

  Funcţia al cărui zero trebuie calculat împreună cu structura sa de derivare se fixează într-o clasă ce implementează interfaţa `Univariate DifferentiableFunction.` [1]. Pachetul *apache commons-math* implementează metoda derivării automate. Metoda nu încearcă să genereze o aproximaţie a derivatei, în schimb utilizează o structură algebrică dublată de o structură de date specifică, denumite structură de derivare

[1] Metoda a fost introdusă de L. B. Rall (1986) iar o prezentare didactică este dată de Kalman D., 2002, *Double Recursive Multivariate Automatic Differentiation.* Mathematics Magazine, 75, no. 3,187-202.

(*derivative structure*). Calculele se efectuează utilizând reprezentarea obişnuită a datelor - în dublă precizie. Astfel va exista o eroare de rotunjire, dar nu va exista o eroare de metodă.

```
import org.apache.commons.math3.analysis.solvers.NewtonRaphsonSolver;
import org.apache.commons.math3.analysis.UnivariateFunction;
import org.apache.commons.math3.analysis.differentiation.
  UnivariateDifferentiableFunction;
import org.apache.commons.math3.analysis.differentiation.
  DerivativeStructure;


class Functia implements UnivariateDifferentiableFunction{
  public DerivativeStructure value(DerivativeStructure x){
    DerivativeStructure t1=x.multiply(Math.log(2)).exp();
    DerivativeStructure t2=x.pow(2);
    return new DerivativeStructure(1,t1,-1,t2);
  }

  public double value(double x){
    return Math.exp(x*Math.log(2))-x*x;
  }
}

public class MetodaTangentei{
  public static void main(String[] args){
    UnivariateDifferentiableFunction function = new Functia();
    double absoluteAccuracy=1.0e-5;
    NewtonRaphsonSolver solver=
      new NewtonRaphsonSolver(absoluteAccuracy);
    int maxEval=1000;
    double min=-0.8;
    double max=-0.5;
    try{
      double c = solver.solve(maxEval,function,min,max);
      System.out.println("Solutia : "+c);
    }
    catch(Exception e){
      System.out.println("Exception : "+e.getMessage());
    }
  }
}
```

- Rezolvarea unei ecuaţii polinomiale utilizând metoda Laguerre. În acest caz se calculează toate rădăcinile reale şi / sau complexe ale unui polinom cu coeficienţi reali sau complecşi.

Exemplul 3.1.2 *Să se calculeze rădăcinile polinomului*

$$(1+x+x^2)^2(x-1-2i) = x^5+(1-2i)x^4+(1-4i)x^3+(-1-6i)x^2+(-1-4i)x-1-2i.$$

```
import org.apache.commons.math.analysis.solvers.LaguerreSolver;
import org.apache.commons.math.analysis.polynomials.PolynomialFunction;
import org.apache.commons.math.complex.Complex;

public class MetodaLaguerre{
  public static void main(String[] args){
    double[] dummyCoeff={1,1};
    PolynomialFunction polinom=new PolynomialFunction(dummyCoeff);
    LaguerreSolver solver = new LaguerreSolver(polinom);
```

```
10      Complex initial=new Complex(1,1);
11      double[] realCoeff={-1,-1,-1,1,1,1};
12      double[] imagCoeff={-2,-4,-6,-4,-2,0};

14      int grad=realCoeff.length;
15      Complex[] complexCoeff=new Complex[grad];
16      for(int i=0;i<grad;i++)
17        complexCoeff[i]=new Complex(realCoeff[i],imagCoeff[i]);
18      try{
19        Complex[] roots = solver.solveAll(complexCoeff,initial);
20        System.out.println("Radacinile : ");
21        for(int i=0;i<grad;i++)
22          System.out.println(roots[i].getReal()+" +I "+
23            roots[i].getImaginary());
24      }
25      catch(Exception e){}
26    }
27  }
```

Rezultatele obţinute sunt

```
Radacinile :
-0.49999991076354433 +I 0.8660254776593728
-0.49999980587362236 +I -0.8660253012543757
1.0000000000000082 +I 2.000000000000006
-0.5000000892365055 +I 0.8660253299095184
-0.5000001941263359 +I -0.8660255063145212
```

- Calculul valorii unei funcţii de interpolare. Sunt implementate interpolarea cu funcţie spline cubică naturală (valoarea derivatei de ordinul doi în punctele de interpolare de abscisă minimă şi maximă este 0), interpolarea Lagrange (algoritmul Neville, prin reprezentarea polinomului de interpolare cu diferenţe divizate).

Exemplul 3.1.3 *Să se calculeze valoarea în 0.5 a funcţiei spline cubice naturală de interpolare a funcţiei $f(x) = |x|$ utilizând nodurile -2,-1,0,1,2.*

```
1  import
2    org.apache.commons.math.analysis.interpolation.SplineInterpolator;
3  import
4    org.apache.commons.math.analysis.interpolation.UnivariateRealInterpolator;
5  import org.apache.commons.math.analysis.UnivariateRealFunction;

7  public class InterpolareSpline{
8    public static void main(String[] args){
9      double x[]={-2,-1,0,1,2};
10     double y[]={2,1,0,1,2};
11     try{
12       UnivariateRealInterpolator interpolator=new SplineInterpolator();
13       UnivariateRealFunction function=interpolator.interpolate(x,y);
14       double t=0.5;
15       double z=function.value(t);
16       System.out.println("Valoarea interpolata in "+t+" este "+z);
17     }
18     catch(Exception e){
19       System.out.println("Exception : "+e.getMessage());
20     }
21   }
22 }
```

- Integrare numerică. Se poate utiliza metoda trapezelor, metoda Simpson sau metoda Romberg (extrapolare tip Richardson pornind de la metoda trapezelor).

Exemplul 3.1.4 *Să se calculeze* $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x = \frac{\pi}{8}\ln 2 \approx 0.2721983$.

Utilizând metoda lui Simpson (1.2), codul calculului este

```
1  import org.apache.commons.math.analysis.integration.SimpsonIntegrator;
2  import org.apache.commons.math.analysis.UnivariateRealFunction;

4  class Functia implements UnivariateRealFunction{
5      public double value(double x){
6        return Math.log(1+Math.tan(x));
7    }
8  }

10 public class MetodaSimpson{
11   public static void main(String[] args){
12     UnivariateRealFunction function = new Functia();
13     try{
14       UnivariateRealIntegrator integrator=
15         new SimpsonIntegrator(function);
16       double integrala=integrator.integrate(0,Math.PI/4);
17       System.out.println("Integrala : "+integrala);
18     }
19     catch(Exception e){
20       System.out.println("Exception : "+e.getMessage());
21     }
22   }
23 }
```

- Transformarea Fourier discretă.

Exemplul 3.1.5 *Calculul coeficienţilor Fourier utilizând transformarea Fourier discretă.*

Fie $\mathbb{C}_n$ mulţimea şirurilor de numere complexe, periodice cu perioada n :

$$\mathbb{C}_n = \{x = (x_k)_{k\in\mathbb{Z}} : x_k \in \mathbb{C}, x_k = x_{k+n}, \forall k \in \mathbb{Z}\}.$$

Transformarea Fourier discretă este un operator liniar $F : \mathbb{C}_n \to \mathbb{C}_n$ definit prin

$$y = F(x), \quad x = (x_k)_{0\le k\le n-1} \quad y = (y_k)_{0\le k\le n-1}$$

$$y_k = \sum_{j=0}^{n-1} x_j w^{-kj}, \qquad 0 \le k \le n-1, \tag{3.1}$$

unde $w = e^{i\frac{2\pi}{n}}$. Şirul y se numeşte transformata Fourier discretă a şirului x.

Dacă $f : \mathbb{R} \to \mathbb{R}$ o funcţie continuă şi periodică cu perioada 2π, atunci are loc dezvoltarea în serie Fourier

$$f(x) = \frac{a_0}{2} + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx) \tag{3.2}$$

având coeficienţii

$$a_0 = \frac{1}{\pi}\int_0^{2\pi} f(x)dx \qquad a_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\cos kxdx \qquad b_k = \frac{1}{\pi}\int_0^{2\pi} f(x)\sin kxdx$$

pentru $k \in N^*$. Coeficienţii Fourier complecşi se definesc prin

$$c_k = \frac{a_k - ib_k}{2} = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}dx.$$

Aproximăm integrala de mai sus cu formula trapezelor. Dacă $n \in N^*$ este parametrul de discretizare atunci se obţine

$$c_k \approx \frac{1}{n}\sum_{j=0}^{n-1} f(\frac{2\pi}{n}j)e^{-ik(\frac{2\pi}{n}j)} = \frac{1}{n}\sum_{j=0}^{n-1} f(\frac{2\pi}{n}j)w^{-jk}. \tag{3.3}$$

Astfel, şirul $c = (c_k)_{0\le k\le n-1}$ este aproximat de $\frac{1}{n}F_n(y)$, unde $y = (y_j)_{0\le j\le n-1}, y_j = f(\frac{2\pi}{n}j)$.

```
1  import org.apache.commons.math.complex.Complex;
2  import org.apache.commons.math.analysis.UnivariateRealFunction;
3  import org.apache.commons.math.transform.FastFourierTransformer;
4  import java.text.DecimalFormat;

6  class Functia implements UnivariateRealFunction{
7     public double value(double x){
8        return Math.sin(x)+Math.cos(3*x);
9     }
10 }

12 public class CoeficientiFourier{
13    public static void main(String[] args){
14       int n=16;
15       double eps=1e-15;
16       UnivariateRealFunction function = new Functia();
17       double[] fVal=new double[n];
18       double[] a=new double[n];
19       double[] b=new double[n];
20       Complex[] fft=new Complex[n];
21       DecimalFormat f=new DecimalFormat("0.0000E0");
22       try{
23          for(int i=0;i<n;i++)
24             fVal[i]=function.value(2*i*Math.PI/n);
25          FastFourierTransformer transformer=new FastFourierTransformer();
26          fft=transformer.transform(fVal);
27          for(int i=0;i<n;i++){
28             a[i]=2*fft[i].getReal()/n;
29             if(Math.abs(a[i])<eps) a[i]=0;
30             b[i]=-2*fft[i].getImaginary()/n;
31             if(Math.abs(b[i])<eps) b[i]=0;
32          }
33          System.out.println("Coeficientii Fourier : ");
34          for(int i=0;i<n/2;i++)
35             System.out.println("a["+i+"] = "+f.format(a[i])+
36                "    b["+i+"] = "+f.format(b[i]));
37       }
38       catch(Exception e){
39          System.out.println("Exception : "+e.getMessage());
40       }
41    }
42 }
```

Rezultatele obţinute sunt

```
Coeficientii Fourier :
a[0] = 0.0000E0   b[0] = 0.0000E0
a[1] = 0.0000E0   b[1] = 1.0000E0
a[2] = 0.0000E0   b[2] = 0.0000E0
a[3] = 1.0000E0   b[3] = 0.0000E0
a[4] = 0.0000E0   b[4] = 0.0000E0
a[5] = 0.0000E0   b[5] = 0.0000E0
a[6] = 0.0000E0   b[6] = 0.0000E0
a[7] = 0.0000E0   b[7] = 0.0000E0
```

- Rezolvarea problemelor cu condiţii iniţiale pentru ecuaţii şi sisteme de ecuaţii diferenţiale ordinare.

```
1  import org.apache.commons.math.ode.FirstOrderDifferentialEquations;
2  import org.apache.commons.math.ode.nonstiff.ClassicalRungeKuttaIntegrator;
3  import org.apache.commons.math.ode.FirstOrderIntegrator;
4  import org.apache.commons.math.ode.sampling.FixedStepHandler;
5  import org.apache.commons.math.ode.sampling.StepNormalizer;
6  import org.apache.commons.math.ode.DerivativeException;

8  class Problema implements FirstOrderDifferentialEquations{
9    public void computeDerivatives(double t, double[]y, double[]yDot){
10     yDot[0]=y[0];
11   }

13   public int getDimension(){
14     return 1;
15   }
16 }

18 public class MetodaRungeKutta{
19   static int k=-1;
20   static double[][] z=new double[101][2];

22   public static void main(String[] args){
23     Problema ode=new Problema();
24     double pas=0.01;
25     int n=101;
26     ClassicalRungeKuttaIntegrator integrator =
27        new ClassicalRungeKuttaIntegrator(pas);
28     double t0=0;
29     double[] y0={1};
30     double t=n*pas;
31     double[] y=new double[y0.length];

33     FixedStepHandler stepHandler0 = new FixedStepHandler(){
34        public void handleStep(double t, double[] y,
35           double[] yDot, boolean isLast)throws DerivativeException{
36             k++;
37             z[k][0]=t;
38             z[k][1]=y[0];
39           }
40     };
41     StepNormalizer stepHandler=new StepNormalizer(pas,stepHandler0);
42     integrator.addStepHandler(stepHandler);
43     try{
44       integrator.integrate(ode,t0,y0,t,y);
45       for(int i=0;i<n;i++){
46          System.out.println(z[i][0]+" "+z[i][1]);
47       }
48     }
49     catch(Exception e){
50       System.out.println("Exception : "+e.getMessage());
51     }
```

```
52     }
53 }
```

3.2 *Jama*

Jama - ***Ja**va **ma**trix* conţine clase pentru rezolvarea unor probleme de algebră liniară (matrice; factorizarea LU, QR, Cholesky; valori proprii, descompunerea valorii singulare).

Exemplul 3.2.1 *Să se rezolve sistemul algebric de ecuaţii liniare*

$$\begin{cases} x_1 + 2x_2 + 3x_3 + 4x_4 = 11 \\ 2x_1 + 3x_2 + 4x_3 + x_4 = 12 \\ 3x_1 + 4x_2 + x_3 + 2x_4 = 13 \\ 4x_1 + x_2 + 2x_3 + 3x_4 = 14 \end{cases}$$

Pentru rezolvarea unui sistem $Ax = b$, metoda `solve` a clasei `Matrix` se poate folosi doar în cazul matricei A nesingulare. Codul rezolvării este extrem de simplu:

```
1  import Jama.*;
2  import java.io.*;
3  import java.text.*;

5  public class SistemLiniar{
6    public static void main(String args[]){
7      double[][] a={{1,2,3,4},{2,3,4,1},{3,4,1,2},{4,1,2,3}};
8      double[] b={11,12,13,14};
9      int n=a.length;

11     Matrix A=new Matrix(a);
12     Matrix B=new Matrix(b,n);
13     Matrix X=A.solve(B);
14     System.out.println("Solutia");
15     X.print(NumberFormat.getNumberInstance(),10);
16   }
17 }
```

Exemplul 3.2.2 *Să se determine factorizarea LU a matricei*

$$A = \begin{pmatrix} 1 & 2 & -1 & 3 & 2 \\ 2 & 4 & -2 & 5 & 1 \\ -1 & -2 & 1 & -3 & -4 \\ 3 & 6 & 2 & 10 & 7 \\ 1 & 2 & 4 & 0 & 4 \end{pmatrix}.$$

Din nou codul este explicit:

```
1  import Jama.*;
2  import java.io.*;
3  import java.text.*;

5  public class LUfact{
6    public static void main(String args[]){
7      double [][] a={{1,2,-1,3,2},{2,4,-2,5,1},{-1,-2,1,-3,-4},
8        {3,6,2,10,7},{1,2,4,0,4}};
```

```
9       int n=a.length;

11      Matrix l,u,p,v;
12      int [] piv;
13      Matrix m=new Matrix(a);
14      LUDecomposition lu=m.lu();
15      l=lu.getL();
16      u=lu.getU();
17      piv=lu.getPivot();
18      System.out.println("Matricea L");
19      l.print(NumberFormat.getNumberInstance(),10);
20      System.out.println("Matricea U");
21      u.print(NumberFormat.getNumberInstance(),10);
22      System.out.println("Matricea P");
23      p=new Matrix(n,n);
24      for(int i=0;i<n;i++)
25        p.set(i,piv[i],1);
26      p.print(NumberFormat.getNumberInstance(),10);

28      System.out.println("Verificare : P A - L U = 0 (?)");
29      v=new Matrix(n,n);
30      v=p.times(m).minus(l.times(u));
31      v.print(NumberFormat.getNumberInstance(),10);
32    }
33  }
```

cu rezultatele

```
Matricea L

         1         0         0         0         0
     0.667         1         0         0         0
     0.333         0         1         0         0
     0.333         0      -0.5         1         0
    -0.333         0       0.5        -1         1

Matricea U

         3         6         2        10         7
         0         0    -3.333    -1.667    -3.667
         0         0     3.333    -3.333     1.667
         0         0         0        -2       0.5
         0         0         0         0        -2

Matricea P

         0         0         0         1         0
         0         1         0         0         0
         0         0         0         0         1
         1         0         0         0         0
         0         0         1         0         0

Verificare : P A - L U = 0 (?)

         0         0         0         0         0
         0         0        -0         0         0
         0         0         0         0         0
         0         0        -0         0         0
         0         0         0         0         0
```

Capitolul 4

Calcul simbolic în Java

Cel mai simplu mod de efectuare a unor calcule simbolice într-un program Java este prin intermediul produselor *Mathematica, Maple.*

Symja - Java Computer Algebra Library este o bibliotecă de pachete Java pentru calcul simbolic (`https://bitbucket.org/axelclk/symja_android_library/wiki/Home`) dar şi numeric.

4.1 Calcul simbolic prin *Mathematica*

Structura unui program Java care apelează o funcţie *Mathematica* de calcul simbolic este cea prezentată în 2.1.

Apelarea şi generarea rezultatului sub forma uzuală din consola *Mathematica* se obţine cu funcţia clasei `com.wolfram.jlink.KernelLink`

`evaluateToOutputForm(String` *apelMathematica,* `0)`

Pentru descompunerea în factori a expresiei algebrice $(x+y)^7-x^7-y^7$ codul de apelare a funcţiei `Factor` este

```
1  import com.wolfram.jlink.*;
2  public class Factor{
3    public static void main(String[] args){
4      KernelLink ml=null;
5      try{
6        String[] mlArgs = {"-linkmode", "launch", "-linkname", args[0]};
7        ml = MathLinkFactory.createKernelLink(mlArgs);
8      }
9      catch(MathLinkException e){
10       System.out.println("Fatal opening link error : "+e.getMessage());
11       System.exit(1);
12     }
13     try{
14       ml.discardAnswer();
15       String result=ml.evaluateToOutputForm("Factor[(x+y)^7-x^7-y^7]",0);
16       System.out.println("Factor[(x+y)^7-x^7-y^7]");
17       System.out.println(result);
18     }
19     catch(Exception e){
20       System.out.println("MathLinkException : "+e.getMessage());
21     }
```

```
22      finally{
23        ml.close();
24      }
25    }
26  }
```

Rezultatul va fi

$\text{Factor}[(x+y)^7 - x^7 - y^7]$

$7xy(x+y)(x^2+xy+y^2)^2$

Analog

- integrala $\int_0^{\frac{\pi}{4}} \ln(1+\tan x)\mathrm{d}x$ se calculează cu funcţia `Integrate[Log[1+Tan[x]],x, 0,Pi/4]` din *Mathematica.* Se obţine: $\frac{\text{Pi Log[2]}}{8}$.

- problema cu valoare iniţială $\dot{y}+y\tan x = \frac{1}{\cos x}$, $y(0)=1$ se calculează cu `DSolve[y'[x] +y[x] Tan[x]==1/Cos[x],y[0]==1,y[x],x]`. Rezultatul este $\{\{\text{y[x]} \to \text{Cos[x]+Sin[x]}\}\}$.

4.2 Calcul simbolic prin *Maple*

Şablonul de programare coincide cu cel prezentat în secţiunea 2.2. Pentru descompunerea în factori a expresiei algebrice $(x+y)^7 - x^7 - y^7$ codul este

```
1  import com.maplesoft.openmaple.*;
2  import com.maplesoft.externalcall.MapleException;
3  class Factor{
4    public static void main( String args[] ){
5      try{
6        String[] argsMaple={"java"};
7        Engine engine=new Engine(argsMaple,new EngineCallBacksDefault(),null,null);
8        Algebraic sol=engine.evaluate("factor((x+y)^7-x^7-y^7);");
9      }
10     catch (MapleException e){
11       System.out.println("MapleException : "+e.getMessage());
12       return;
13     }
14   }
15 }
```

Rezultatul este `7*x*y*(x+y)*(x^2+x*y+y^2)^2`

Dacă expresia de evaluat este problema cu valoare iniţială

*dsolve(diff(y(x),x)+y(x)*tan(x)-1/cos(x)=0,y(0)=1,y(x));*

atunci se obţine `y(x)=cos(x)*tan(x)+cos(x)` . Pentru expresia

*simplify(dsolve(diff(y(x),x)+y(x)*tan(x)-1/cos(x)=0,y(0)=1,y(x)));*

se va obţine `y(x)=sin(x)+cos(x)` .

4.3 *Symja - Java Computer Algebra Library*

Resursele necesare variantei curente sunt:

COMMONS_MATH4_SYMJA.jar
symja-*.jar
log4j-1.2.17.jar

Sintaxa utilizată este cea din *Mathematica*, dar în editarea expresiilor de calcul nu se face distincţie între literele mari şi mici iar parantezele pătrate pot fi înlocuite cu paranteze rotunde. Astfel expresiile

```
Factor[(x+y)^7-x^7-y^7]
factor[(x+y)^7-x^7-y^7]
factor((x+y)^7-x^7-y^7)
```

sunt echivalente.

Trebuie subliniat că posibilităţile de calcul simbolic nu coincid cu cele din *Mathematica.*
Şablonul de programare este

```
ExprEvaluator util = new ExprEvaluator();
IExpr result=util.evaluate("Expresia de calcul");
System.out.println(result.toString());
```

În exemplul următor se calculează:

1. descompunerea în factori a expresiei $(x+y)^7 - x^7 - y^7$;
2. $\frac{\mathrm{d}\sin 2x \cos x}{\mathrm{d}x}$;
3. $\frac{\mathrm{d}^2 \arctan x}{\mathrm{d}x^2}$;
4. $\int \frac{1}{(1+x^4)} \mathrm{d}x$;
5. $\int_0^\infty \frac{1}{1+x^2)} \mathrm{d}x$;

cu codul Java

```
1  import org.matheclipse.core.eval.ExprEvaluator;
2  import org.matheclipse.core.interfaces.IExpr;

4  public class ESymja{
5    public static void main(String[] args) {
6      try {
7        ExprEvaluator util = new ExprEvaluator();
8        IExpr result=util.evaluate("Factor[(x+y)^7-x^7-y^7]");
9        System.out.println(result.toString());

11       result = util.evaluate("D[Sin[2 x] Cos[x],x]");
12       System.out.println(result.toString());

14       result = util.evaluate("D[ArcTan[x],{x,2}]");
15       System.out.println(result.toString());

17       result = util.evaluate("Integrate[1/(1+x^4),x]");
18       System.out.println(result.toString());

20       result = util.evaluate("Integrate[1/(1+x^2),{x,0,Infinity}]");
21       System.out.println(result.toString());
```

```
22      }
23      catch (Exception e) {
24        e.printStackTrace();
25      }
26    }
27  }
```

Rezultatele sunt, respectiv

```
1.
 7*(x+y)*x*y*(x^2+x*y+y^2)^2
2.
 2*Cos(x)*Cos(2*x)-Sin(x)*Sin(2*x)
3.
 (-2*x)/(1+x^2)^2
 1/2*(-ArcTan(1+(-2*x)/Sqrt(2))/Sqrt(2)+ArcTan(1+(2*x)/Sqrt(2))/Sqrt(2))+
   1/2*(Log(1+x*Sqrt(2)+x^2)/(2*Sqrt(2))-Log(1-x*Sqrt(2)+x^2)/(2*Sqrt(2)))
4.
 Pi/2
```

Capitolul 5

Expresie de calcul dată ca `String`

Execuţia unui program de calcul ştiinţific poate solicita furnizarea de către client a unor expresii de calcul. Acestea pot reprezenta membrul stâng al unei ecuaţii algebrice $f(x) = 0$ sau membrul drept al unei ecuaţii diferenţiele ordinare $\dot{x} = f(x)$, etc. Expresia de calcul, notată prin $f(x)$, poate conţine simboluri matematice uzuale, chiar nume de funcţii.

String-ul introdus de client trebuie interpretat de programul de calcul ca o expresie de evaluat, chiar permiţând evaluarea ei pentru tot felul de valori date parametrilor.

Scopul acestui capitol este prezentarea de soluţii pentru aceste probleme.

Vom utiliza produsele informatice:

- *Java Expression Parser - JEP;*
- *MathEclipse Parser.*

În afara acestora semnalăm existenţa urmatoarelor softuri *exp4j, javaluator* cât şi posibilitatea utilizării motorului *Javascript* din Java.

5.1 *Java Expression Parser* - JEP

JEP este probabil produsul cel mai reprezentativ pentru problema enunţată. Până la versiunea 2.4.1, *JEP* a fost un produs gratuit, versiunile următoare fiind produse comerciale. Astfel, ne vom limita doar la versiunea gratuită 2.4.1.

Variabila de sistem `classpath` trebuie să conţină referinţa către `jep-2.4.1.jar`.

Utilizarea produsului. Evaluarea unei expresii în care intervin variabile se obţine prin

1. Generarea unui *convertor JEP*

```
JEP parser=new JEP();
```

2. Definirea variabilelor. Dacă variabila *var* are valoarea *val* definirea ei se face prin

```
parser.addVariable(var,val);
```

3. Definirea expresiei / formulei de evaluat. Expresia de calcul dată de stringul *s_exp* se introduce prin

```
parser.parseExpression(s_exp);
```

4. Rezultatul evaluării se obţine prin

```
parser.getValue();
```

Operaţiile aritmetice se indică în mod obişnuit prin

adunare	+	scădere	-
înmulţire	*	împărţire	/
ridicare la putere	^		

Dacă am dori să calculăm expresia $x^2 + y^2$ pentru $x = 3$ şi $y = 4$, atunci codul este

```
1  import org.nfunk.jep.*;
2
3  class TestJep1{
4    public static void main(String args[]){
5      double x=3,y=4;
6      String s_exp="x^2+y^2";
7      JEP parser=new JEP();
8      parser.addVariable("x",x);
9      parser.addVariable("y",y);
10     parser.parseExpression(s_exp);
11     double rezultat=parser.getValue();
12     System.out.println("Rezultat: "+rezultat);
13   }
14 }
```

Funcţiile uzuale recunoscute de *JEP* sunt:

$\sin(x)$	$\sin(x)$	$\arcsin(x)$	$\text{asin}(x)$
$\cos(x)$	$\cos(x)$	$\arccos(x)$	$\text{acos}(x)$
$\text{tg}(x)$	$\tan(x)$	$\text{arctg}(x)$	$\text{atan}(x)$
$\text{sh}(x)$	$\sinh(x)$	$\text{arcsh}(x)$	$\text{asinh}(x)$
$\text{ch}(x)$	$\cosh(x)$	$\text{arcch}(x)$	$\text{acosh}(x)$
$\text{th}(x)$	$\tanh(x)$	$\text{arcth}(x)$	$\text{atanh}(x)$
e^x	$\exp(x)$	$\ln(x)$	$\ln(x)$
$\lg(x)$	$\log(x)$	$\lvert x \rvert$	$\text{abs}(x)$
$\sqrt{x}$	$\text{sqrt}(x)$		

Constantele recunoscute de *JEP* sunt:

e	e	π	pi	$i = \sqrt{-1}$	i

Recunoaşterea acestor funcţii şi constante presupune declararea lor:

```
parser.addStandardFunctions();
parser.addStandardConstants();
```

Calculul expresiei $\sin\frac{\pi}{4}+\sin\frac{\pi}{3}$ este

```
double x=Math.PI/3;
double y=Math.PI/4;
String s_exp="sin(x)+sin(y)";
JEP parser=new JEP();
parser.addStandardFunctions();
parser.addStandardConstants();
parser.addVariable("x",x);
parser.addVariable("y",y);
parser.parseExpression(s_exp);
double rezultat=parser.getValue();
```

Colecţia de funcţii recunoscute de *JEP* se poate extinde. Exemplificăm cu includerea funcţiei $\mathrm{erf}(x)=\frac{2}{\pi}\int_0^x e^{-t^2}\mathrm{d}t$, pe care o preluăm din pachetul *org.apache.commons.math.*

În acest sens, şablonul de programare solicită definirea unei clase de forma

```
1  import java.util.*;
2  import org.nfunk.jep.*;
3  import org.nfunk.jep.function.*;
4  import org.apache.commons.math3.special.*;

6  class JepErf extends PostfixMathCommand {
7    public JepErf() {
8      numberOfParameters = 1;
9    }

11   public void run(Stack inStack) throws ParseException{
12     checkStack(inStack);
13     Object param = inStack.pop();
14     if (param instanceof Double){
15       try{
16         double r=0,x=((Double)param).doubleValue();
17         if(Math.abs(x)<26)
18           r = Erf.erf(x);
19         else{
20           if(x>0)
21             r=1;
22           else
23             r=-1;
24         }
25         inStack.push(new Double(r));
26       }
27       catch(Exception e){
28         throw new ParseException(e.getMessage());
29       }
30     }
31     else{
32       throw new ParseException("Invalid parameter type");
33     }
34   }
35 }
```

În vederea utilizării, noua funcţie se declară prin

```
parser.addFunction("erf", new JepErf());
```

erf reprezintă numele sub care se recunoaşte funcţia într-un string, iar *JepErf* este numele clasei ce evaluează funcţia. În urma acestei operaţii, funţia *erf* va putea fi evaluată, la fel ca oricare altă funcţie.

Calculul valorii $\mathrm{erf}(10)\approx 1$ constă din

```
double x=10;
String s_exp="erf(x)";
JEP parser=new JEP();
parser.addStandardFunctions();
parser.addStandardConstants();
parser.addFunction("erf", new JepErf());
parser.addVariable("x",x);
parser.parseExpression(s_exp);
double rezultat=parser.getValue();
```

5.2 *MathEclipse-Parser*

MathEclipse-Parser este un produs gratuit care evaluează numeric expresiile de calcul date ca string şi în plus poate fi încorporat ca *modul* în *Google Web Toolkit* (GWT).

A doua modalitate de folosire este motivul pentru care ne-am oprit asupra acestui produs informatic.

În cele ce urmează, vom ilustra câteva facilităţi ale lui *MathEclipse-Parser.* Se utilizează fişierul descărcat `matheclipse-parser-*.jar,` iar pentru includerea în GWT acesta trebuie dezarhivat.

MathEclipse-Parser oferă posibilitatea de lucru cu variabile reale - de tip predefinit *double* dar şi cu variabile complexe.

Prelucrarea expresiilor reale

Maniera de lucru este asemănătoare cu cea din *JEP:*

1. Generarea unui motor de evaluare (*parser*).

   ```
   DoubleEvaluator engine = new DoubleEvaluator();
   ```

2. Definirea variabilelor.

   ```
   IDoubleValue vx = new DoubleVariable(val);
   engine.defineVariable("x",vx);
   ```

3. Definirea expresiei / formulei de evaluat. Expresia de calcul dată de stringul *s_exp* conţine simbolul x.

   Bineînţeles, se pot defini atâtea variabile de câte este nevoie.

   Numele funcţiilor coincid cu cele din *Mathematica*, iar argumentele se scriu între paranteze drepte.

4. Rezultatul evaluării expresiei *s_exp* se obţine prin

   ```
   double rezultat = engine.evaluate(s_expr);
   ```

Constantele recunoscute de *MathEclipse-Parser* sunt:

e	E	π	Pi

Astfel evaluarea expresiei $x^2 + y^2$ pentru $x = 3$ şi $y = 4$, are codul

```
double x=3,y=4;
String s_expr="x^2+y^2";
IDoubleValue vx = new DoubleVariable(x);
IDoubleValue vy = new DoubleVariable(y);
DoubleEvaluator engine = new DoubleEvaluator();
engine.defineVariable("x",vx);
engine.defineVariable("y",vy);
double rezultat = engine.evaluate(s_expr);
```

iar calculul expresiei $\sin\frac{\pi}{4} + \sin\frac{\pi}{3}$ este

```
String s_expr="Sin[Pi/3]+Sin[Pi/4]";
DoubleEvaluator engine = new DoubleEvaluator();
double rezultat = engine.evaluate(s_expr);
```

Prelucrarea expresiilor complexe

Analiza următorului program, scoate în evidenţa posibilităţile oferite, cât şi maniera de programare

```
1  import org.matheclipse.parser.client.eval.*;
2  import org.matheclipse.parser.client.math.*;
3  public class ExpresiiComplexe{

5    public static void main(String[] args){
6      try {
7        Complex z=new Complex(1,3);
8        System.out.print("z=");
9        System.out.println(ComplexEvaluator.toString(z));

11       System.out.println("Conjugatul");
12       Complex w=z.conjugate();
13       System.out.println(ComplexEvaluator.toString(w));

15       System.out.println("Suma");
16       System.out.println(ComplexEvaluator.toString(z.add(w)));

18       System.out.println("Produsul");
19       System.out.println(ComplexEvaluator.toString(z.multiply(w)));

21       System.out.println("Catul");
22       System.out.println(ComplexEvaluator.toString(z.divide(w)));

24       System.out.println("Modulul");
25       System.out.println(z.abs());

27       System.out.println("\nFunctii complexe");
28       System.out.println("sin(z)");
29       System.out.println(ComplexEvaluator.toString(z.sin()));

31       System.out.println("cos(z)");
32       System.out.println(ComplexEvaluator.toString(z.cos()));

34       System.out.println("tan(z)");
35       System.out.println(ComplexEvaluator.toString(z.tan()));

37       System.out.println("sinh(z)");
```

```
38        Complex u=z.sinh();
39        System.out.println(ComplexEvaluator.toString(u));

41        System.out.println("cosh(z)");
42        Complex v=z.cosh();
43        System.out.println(ComplexEvaluator.toString(v));

45        System.out.println("tanh(z)");
46        System.out.println(ComplexEvaluator.toString(z.tanh()));

48        System.out.println("\nEvaluarea expresiilor complexe");
49        System.out.println("cosh^2(z)-sinh^2(z)");
50        ComplexVariable cu = new ComplexVariable(u);
51        ComplexVariable cv = new ComplexVariable(v);
52        ComplexEvaluator engine = new ComplexEvaluator();
53        String expr="v^2-u^2";
54        engine.defineVariable("u",cu);
55        engine.defineVariable("v",cv);
56        Complex r=engine.evaluate(expr);
57        System.out.println(ComplexEvaluator.toString(r));
58      }
59      catch(Exception e) {
60        System.out.println(e.getMessage());
61      }
62    }
63  }
```

Capitolul 6

Aplicaţii cu interfaţă grafică

Ataşăm o interfaţă grafică unor aplicaţii dezvoltate în capitolele anterioare. Prin intermediul interfeţei grafice, un client (utilizator) introduce datele şi primeşte afişat rezultatul problemei.

Programarea interfeţei grafice se poate baza pe resursele oferite de pachetele *javax.swing* şi *javafx* din distribuţia Java.

Utilizarea mediului integrat de programare (*Integrated Development Environment* - IDE) *Netbeans,* `www.netbeans.org`, uşurează mult munca de programare.

6.1 Rezolvarea unei ecuaţii algebrice

Expresia funcţiei este introdusă de client sub forma unui string. Aplicând rezultatele anterioare, extindem mini-biblioteca prezentată în primul capitol cu clase de tip *DataIn* care implementează corespunzător metoda *public double fct(double x).*

Componenta Java care reţine datele problemei, bazat pe *JEP* (clasa *mathlib.client.ecalg.JepDataIn*), are codul

```
1  package mathlib.client.ecalg;
2  import org.nfunk.jep.*;
3  /**
4   * Extinderea clasei DataIn cu preluarea unor date
5   * ca String-uri care sunt convertite in valori numerice
6   * prin pachetul JEP (Java Expression Parser)
7   */
8  public class JepDataIn extends DataIn{
9    private JEP parser=null;
10   private String var;

12   /**
13    * Constructorul clasei care instantiaza si calibreaza
14    * obiectul JEP utilizat la pentru evaluarea functiei.
15    * @param var Simbolul variabilei.
16    * @param expr Extresia functiei.
17    */
18   public JepDataIn(String var,String expr){
19     this.var=var;
20     parser=new JEP();
21     parser.addStandardFunctions();
22     parser.addStandardConstants();
23     parser.addVariable(var,0);
24     parser.parseExpression(expr);
```

```
25    }
27    /**
28     *  Functia corespunzatoare membrului stang al ecuatiei fct(x)=0.
29     */
30    public double fct(double x){
31       parser.addVariable(var,x);
32       return parser.getValue();
33    }
34 }
```

Reluăm problema rezolvării ecuaţiei algebrice $f(x) = 0$ din 1.2.

6.1.1 Interfaţa grafică bazată pe *JavaFX*

Codul clasei cu interfaţa grafică bazată pe pachetele *javafx* este

```
1  package ecuatiefx;

3  import javafx.application.Application;
4  import javafx.event.ActionEvent;
5  import javafx.scene.Group;
6  import javafx.scene.Scene;

8  import javafx.scene.paint.Color;
9  import javafx.stage.Stage;
10 import javafx.scene.control.Label;
11 import javafx.scene.control.TextField;
12 import javafx.scene.layout.GridPane;
13 import javafx.scene.control.Button;
14 import javafx.event.EventHandler;
15 import javafx.scene.input.MouseEvent;
16 import java.text.DecimalFormat;
17 import mathlib.client.ecalg.*;
18 import mathlib.client.ecalg.impl.MetodaTangenteiWeb;

20 public class Ecuatiefx extends Application {

22     public static void main(String[] args) {
23         Application.launch(Ecuatiefx.class, args);
24     }

26     @Override
27     public void start(Stage primaryStage) {
28         primaryStage.setTitle("Metoda tangentei");
29         Group root = new Group();
30         Scene scene = new Scene(root, 600, 250, Color.LIGHTGREEN);

32         GridPane gridpane=new GridPane();

34         Label labelVar=new Label("Variabila");
35         GridPane.setConstraints(labelVar, 1, 1);
36         Label labelAprox=new Label("Aproximatia initiala");
37         GridPane.setConstraints(labelAprox, 1, 2);
38         Label labelExpr=new Label("Expresia membrului stang");
39         GridPane.setConstraints(labelExpr, 1, 3);
40         Label labelTol=new Label("Toleranta");
41         GridPane.setConstraints(labelTol, 1, 4);
42         Label labelNmi=new Label("Numar maxim de iteratii");
43         GridPane.setConstraints(labelNmi, 1, 5);

45         final TextField TextFieldVar=new TextField();
46         GridPane.setConstraints(TextFieldVar, 2, 1);
47         final TextField TextFieldAprox=new TextField();
48         GridPane.setConstraints(TextFieldAprox, 2, 2);
```

```
49          final TextField TextFieldExpr=new TextField();
50          GridPane.setConstraints(TextFieldExpr, 2, 3);
51          final TextField TextFieldTol=new TextField("1.0e-8");
52          GridPane.setConstraints(TextFieldTol, 2, 4);
53          final TextField TextFieldNmi=new TextField("50");
54          GridPane.setConstraints(TextFieldNmi, 2, 5);

56          Label labelInd=new Label("Indicatorul de raspuns");
57          GridPane.setConstraints(labelInd, 3, 1);
58          Label labelSol=new Label("Solutia");
59          GridPane.setConstraints(labelSol, 3, 2);
60          Label labelVal=new Label("Valoarea in solutie");
61          GridPane.setConstraints(labelVal, 3, 3);
62          Label labelIter=new Label("Numar de iteratii");
63          GridPane.setConstraints(labelIter, 3, 4);

65          final TextField TextFieldInd=new TextField();
66          TextFieldInd.setVisible(false);
67          GridPane.setConstraints(TextFieldInd, 4, 1);
68          final TextField TextFieldSol=new TextField();
69          TextFieldSol.setVisible(false);
70          GridPane.setConstraints(TextFieldSol, 4, 2);
71          final TextField TextFieldVal=new TextField();
72          TextFieldVal.setVisible(false);
73          GridPane.setConstraints(TextFieldVal, 4, 3);
74          final TextField TextFieldIter=new TextField();
75          TextFieldIter.setVisible(false);
76          GridPane.setConstraints(TextFieldIter, 4, 4);

78          Button btn=new Button("Calculeaza");
79          GridPane.setConstraints(btn, 2, 6);
80          btn.setOnAction((new EventHandler<ActionEvent>() {
81              public void handle(ActionEvent me) {
82                  String var=TextFieldVar.getText();
83                  String expr=TextFieldExpr.getText();
84                  DataIn din=new JepDataIn(var,expr);
85                  din.setX((new Double(TextFieldAprox.getText())).doubleValue());
86                  din.setEps((new Double(TextFieldTol.getText())).doubleValue());
87                  din.setNmi((new Integer(TextFieldNmi.getText())).intValue());
88                  IMetodaTangentei obj=new MetodaTangenteiWeb();
89                  DataOut dout=obj.metodaTangentei(din);
90                  DecimalFormat f=new DecimalFormat("0.00000E0");
91                  String sol=f.format(dout.getX());
92                  String val=f.format(dout.getF());
93                  TextFieldInd.setText((new Integer(dout.getInd())).toString());
94                  TextFieldIter.setText((new Integer(dout.getNi())).toString());
95                  TextFieldSol.setText(sol);
96                  TextFieldVal.setText(val);
97                  TextFieldInd.setVisible(true);
98                  TextFieldSol.setVisible(true);
99                  TextFieldVal.setVisible(true);
100                 TextFieldIter.setVisible(true);
101             }
102         }));

104         gridpane.setVgap(8);
105         gridpane.setHgap(8);
106         gridpane.getChildren().
107           addAll(labelVar,labelAprox,labelExpr,labelTol,labelNmi);
108         gridpane.getChildren().
109           addAll(TextFieldVar,TextFieldAprox,TextFieldExpr,TextFieldTol,TextFieldNmi);
110         gridpane.getChildren().
111           addAll(labelInd,labelSol,labelVal,labelIter);
112         gridpane.getChildren().
113           addAll(TextFieldInd,TextFieldSol,TextFieldVal,TextFieldIter);
114         gridpane.getChildren().addAll(btn);
115         root.getChildren().add(gridpane);
```

```
116             primaryStage.setScene(scene);
117             primaryStage.show();
118         }
119 }
```

Imaginea interfeţei grafice completată cu datele problemei rezolvate în 1.2 împreună cu rezultatele obţinute sunt prezentate în Fig. 6.1.

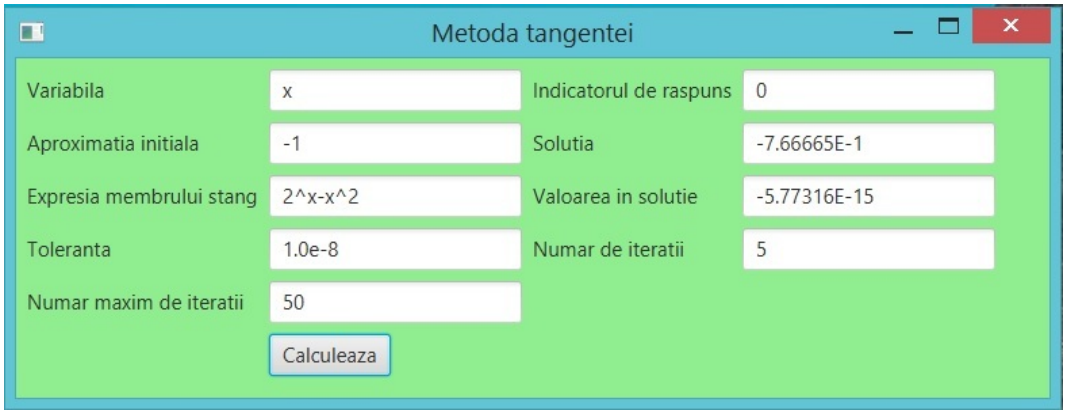


Fig. 6.1: Interfaţa grafică pentru rezolvarea ecuaţiei algebrice.

6.2 Rezolvarea unui sistem algebric de ecuaţii liniare

Rezolvarea unui sistem algebric $Ax = b$ va utiliza extinderea mini-bibliotecii *mathlib,* dezvoltată în 2.3. Rezolvarea se bazează pe funcţia *Scilab* `linsolve`.

6.2.1 Interfaţa grafică bazată pe *Swing*

În *Netbeans,* interfaţa grafică se construieşte aproape în totalitate cu mouse-ul, codul generându-se în fundal. În consecinţă se va prezenta doar codul specific aplicaţiei, codul pentru interfaţa grafică generată de *Netbeans* va fi omis, acest cod este dependent de versiunea utilizată.

Interfaţa grafică este dată în Fig. 6.2.

Clientul editează un fişier text cu coeficienţii sistemului. Fiecare linie a acestui fişier va conţine coeficienţii unei linii a sistemului iar ultimul element va fi termenul liber. Separatorul este caracterul spaţiu (blanc).

După un clic pe butonul *Incarcă fişierul sistemului* va apare fereastra pentru selectarea fişierului cu datele sistemului (Fig. 6.3).

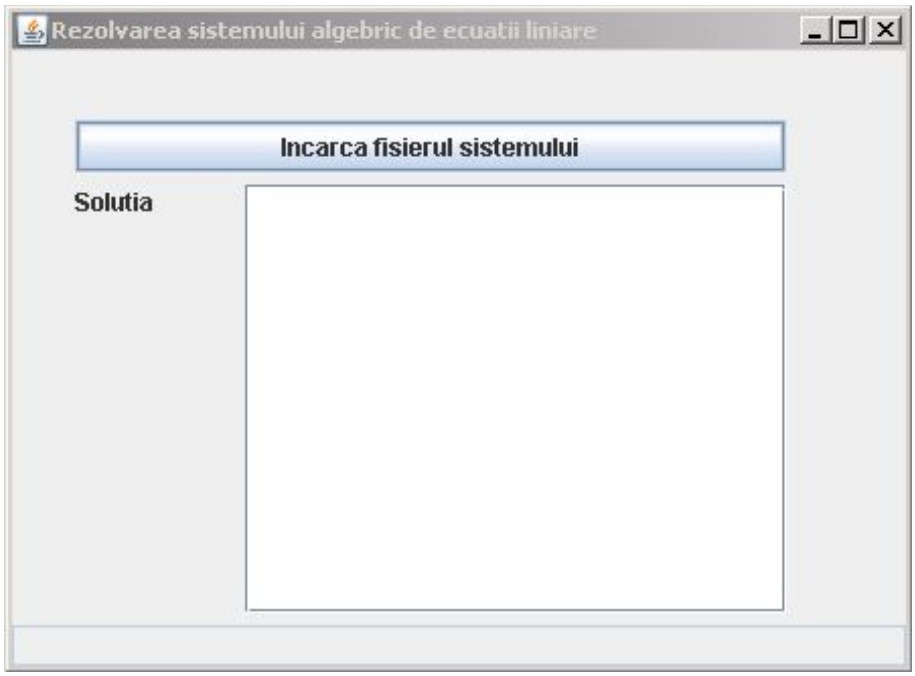


Fig. 6.2: Interfaţa grafică pentru rezolvarea sistemului.

Conţinutul fişierului cu datele sistemului va fi preluat de aplicaţie. Imediat se apelează rezolvarea sistemului algebric de ecuaţii liniare. Rezultatele se afişează într-un control de tip `JTextArea`. Rezultatele din Fig. 6.4 corespund exemplului din 2.3.

Selectarea fişierului se programează utilizând controlul *Swing* `JFileChooser`. Schema de prelucrare poate fi

```
JFileChooser fc=new JFileChooser();
  fc.setDialogTitle(" titlu ");
  fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
  try{
    if (evt.getSource() == jButtonUpload){
      int returnVal = fc.showOpenDialog(this);
      if (returnVal == JFileChooser.APPROVE_OPTION) {
        File file = fc.getSelectedFile();
        // Prelucrarea fisierului
      }
      else{
        throw new Exception("Actiune anulata de client.");
      }
    }
  }
  catch(Exception ex){
    // prelucrarea exceptiei generate
  }
```

Codul sursă (obţinut cu Netbeans) este

```
1 package linear;
2 import javax.swing.JFileChooser;
3 import java.io.*;
```

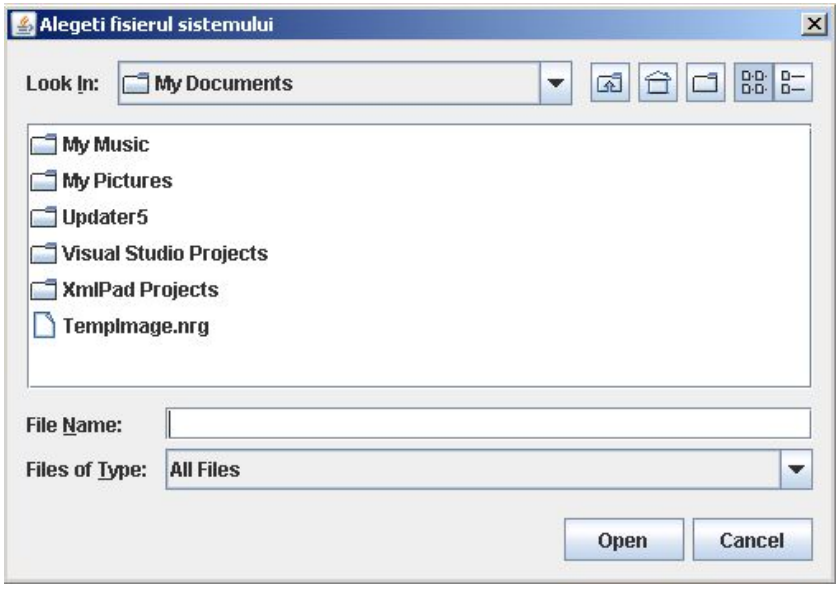


Fig. 6.3: Fereastra de dialog pentru selectarea sistemului.

```
4  import mathlib.client.linear.*;
5  import mathlib.client.linear.impl.RezolvitorScilab;
6  import java.text.DecimalFormat;

8  public class SistemLiniar extends javax.swing.JFrame {

10   public SistemLiniar() {
11       initComponents();
12   }

14   private void initComponents() {
15     // cod generat de Netbeans
16   }

18   private void jButtonUploadMouseClicked(java.awt.event.MouseEvent evt) {
19     jTextFieldStatus.setText("");
20     JFileChooser fc=new JFileChooser();
21     fc.setDialogTitle("Alegeti fisierul sistemului");
22     fc.setFileSelectionMode(JFileChooser.FILES_ONLY);
23     DecimalFormat f=new DecimalFormat("0.0000E0");
24     jTextAreaSol.setText("");
25     try{
26       if (evt.getSource() == jButtonUpload){
27         int returnVal = fc.showOpenDialog(this);
28         if (returnVal == JFileChooser.APPROVE_OPTION) {
29           File file = fc.getSelectedFile();
30           FileInputStream fis=new FileInputStream(file);
31           InputStreamReader isr=new InputStreamReader(fis);
32           BufferedReader br=new BufferedReader(isr);
33           DataIn din=new DataIn();
34           din.setMatrix(br);
35           br.close();
36           isr.close();
37           fis.close();
38           IRezolvitorScilab obj=new RezolvitorScilab();
39           DataOut dout=obj.rezolvitorScilab(din);
40           String rez="";
41           if(dout.isCompatibil()){
42             double[] x=dout.getX();
```

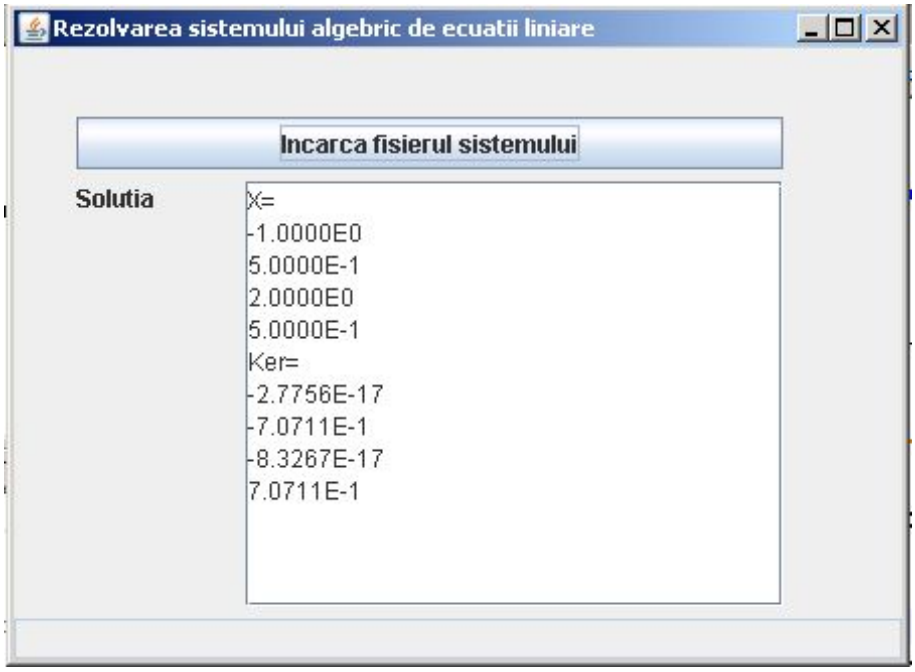


Fig. 6.4: Afişarea rezultatelor

```
43              double[][] k=dout.getK();
44              rez="X=\n";
45              int l=x.length;
46              for(int i=0;i<l;i++){
47                rez+=f.format(x[i]);
48                rez+="\n";
49              }
50              rez+="Ker=\n";
51              if(k!=null){
52                int c=k[0].length;
53                for(int i=0;i<l;i++){
54                  for(int j=0;j<c;j++){
55                    rez+=f.format(k[i][j]);
56                    rez+="      ";
57                  }
58                  rez+="\n";
59                }
60              }
61              else{
62                rez+="[ ]";
63              }
64            }
65            else{
66              rez="Sistem incompatibil !";
67            }
68            jTextAreaSol.setText(rez);
69          }
70          else{
71            throw new Exception("Actiune anulata de client.");
72          }
73        }
74      }
75      catch(Exception ex){
76        jTextFieldStatus.setText(ex.getMessage());
77      }
78    }
```

```
80    public static void main(String args[]) {
81      java.awt.EventQueue.invokeLater(new Runnable() {
82        public void run() {
83          new SistemLiniar().setVisible(true);
84        }
85      });
86    }

88    private javax.swing.JButton jButtonUpload;
89    private javax.swing.JLabel jLabelSol;
90    private javax.swing.JPanel jPanel1;
91    private javax.swing.JScrollPane jScrollPane1;
92    private javax.swing.JTextArea jTextAreaSol;
93    private javax.swing.JTextField jTextFieldStatus;
94  }
```

Codul nereprodus generează interfaţa grafică din Fig. 6.2.

Capitolul 7

Generarea reprezentărilor grafice

Afirmaţia *o imagine grafică spune mai multe decât o mie de cuvinte* este adevărată în numeroase cazuri - şi cu siguranţă când rezultatele corespund valorilor unei funcţii.

Există multe produse informatice care oferă funcţii pentru obţinerea de reprezentări grafice.

7.1 *PtPlot*

O soluţie simplă este dată de produsul *PtPlot,* dezvoltat la Universitatea din California. În cele ce urmează vom utiliza interfaţa de programare oferită (API) pentru generarea graficului unei funcţii reale de variabilă reală. Este nevoie de arhiva *plotapplication.jar* aflată în catalogul *ptolemy\plot* din distribuţie.

Problema pe care dorim să o rezolvă este reprezentarea grafică a unei funcţii reale de variabilă reală.

Datele problemei sunt simbolul variabilei, expresia funcţiei şi intervalul în care se reprezintă funcţia. Acest interval poate să nu fie inclus în domeniul de definiţie al funcţiei. Ideea reprezentării este simplă: în intervalul dat, se consideră o reţea de puncte în care se calculează valorile funcţiei. Generarea graficului este transparentă programatorului, fiind efectuată de *PtPlot.* Rezultatul va fi un obiect `java.awt.image.BufferedImage` a cărei vizualizare cade în sarcina programatorului.

O problemă constă în depistarea punctelor sau intervalelor în care funcţia nu este definită.

Utilizarea resursele pachetului *PtPlot* se declară prin

```
import ptolemy.plot.*;
```

Reprezentarea grafică se va obţine prin intermediul unui obiect

```
Plot plotObj=new Plot();
```

Definim punctele intervalului $[a, b]$ care urmează a fi reprezentate

```
h=(b-a)/n;
int dataset=0;
for(int i=0;i<=n;i++){
  x=a+i*h;
  y=f(x)
```

```
  if((Double.isInfinite(y))||(Double.isNaN(y))){
    dataset++;
  }
  else{
    plotObj.addPoint(dataset,x,y,true);
  }
}
```

Punctele aparţinând unui interval al domeniului de definiţie sunt caracterizate printr-o variabilă întreagă *dataset*. Doar aceste puncte vor fi unite de către *PtPlot*. Acest lucru este indicat prin valoarea `true` atribuită ultimei variabile a metodei `addPoint`.

Reprezentarea grafică este conţinută într-o variabilă de tip `java.awt.image.BufferdImage` şi corespunde unui dreptunghi de rezoluţie $xDim \times yDim$ pixeli.

```
Rectangle rectangle=new Rectangle(xDim,yDim);
BufferedImage image=plotObj.exportImage(rectangle);
```

Metode ale clasei `Plot` (selecţie)

`setTitle(String` *title*`)`	Adaugă titlul reprezentării grafice.
`setXLabel(String` *xLabel*`)`	Adaugă eticheta axei Ox.
`setYLabel(String` *yLabel*`)`	Adaugă eticheta axei Oy.
`setSize(int` *xDim*`,int` *yDim*`)`	Fixează rezoluţia reprezentării.
`setXRange(int` *a*`,int` *b*`)`	Fixează intervalul reprezentării pe axa Ox.
`setYRange(int` *c*`,int` *d*`)`	Fixează intervalul reprezentării pe axa Oy.
`setGrid(boolean` *grid*`)`	Pentru `true` se desenează o reţea de drepte.
`setXLog(boolean` *log*`)`	Pentru `true` axa Ox este logaritmică.
`setYLog(boolean` *log*`)`	Pentru `true` axa Oy este logaritmică.
`setColor(boolean` *color*`)`	Pentru `false` graficul este alb/negru.
`addLegend(int` *dataset*`,String` *legenda*`)`	Adaugă *legenda* setului de date *dataset*.

Redarea într-o fereastră a monitorului se obţine apelând

```
ShowImage si=new ShowImage(image,xDim,yDim);
si.show();
```

Codul programului *ShowImage:*

```
1  package plot2d;
2  import java.awt.*;
3  import javax.swing.JFrame;

5  public class ShowImage{
6    MyCanvas mc=null;
7    int xDim=0;
8    int yDim=0;

10   ShowImage(Image image,int xDim,int yDim){
11     this.xDim=xDim;
12     this.yDim=yDim;
13     mc=new MyCanvas(image);
14   }
15   public void show(){
16     JFrame jframe = new JFrame("Graficul functiei");
17     jframe.addNotify();
18     jframe.getContentPane().setLayout(new BorderLayout());
19     jframe.getContentPane().add(mc,BorderLayout.CENTER);
20     jframe.setSize(xDim,yDim);
21     jframe.setVisible(true);
22   }
23 }
```

unde *MyCanvas* este clasa

```
1  package plot2d;
2  import java.awt.*;

4  public class MyCanvas extends Canvas{
5    Image image=null;

7    MyCanvas(Image image){
8      this.image=image;
9    }
10   @Override
11   public void paint(Graphics g){
12     g.drawImage(image,0,0,this);
13   }
14 }
```

Exemplul 7.1.1 *Program pentru reprezentarea grafică a unei funcţii.*

Datele necesare se introduc prin intermediul unei interfeţe grafice (Fig. 7.1)

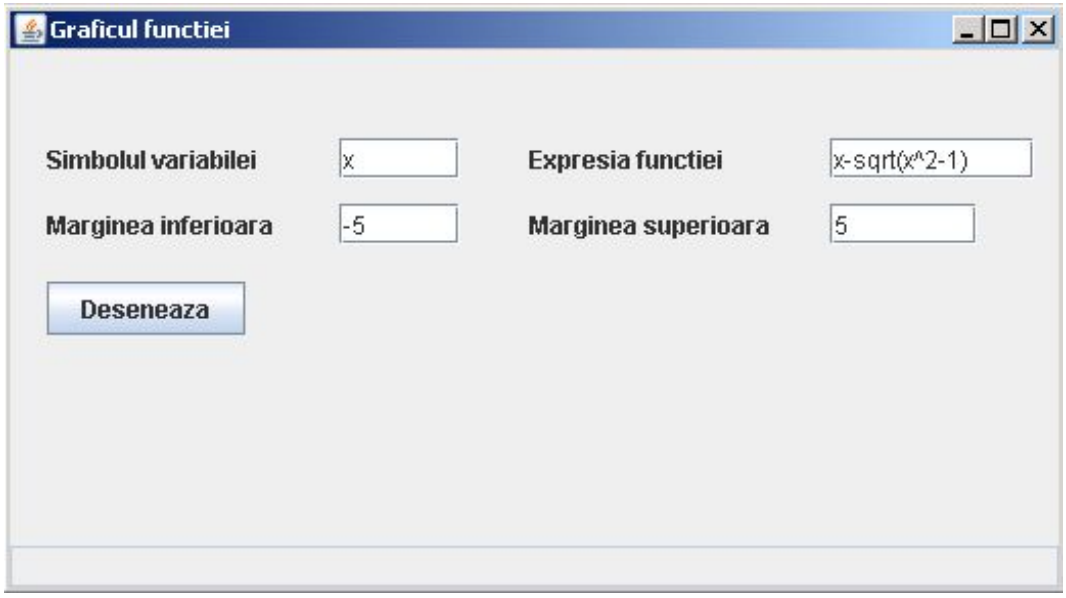


Fig. 7.1: Interfaţa grafică pentru datele graficului unei funcţii.

Codul programului cu interfaţa grafică Swing şi care utilizează *JEP* pentru evaluarea valorilor funcţiei dată ca parametru de tip `String` este

```
1  package plot2d;
2  import ptolemy.plot.*;
3  import org.nfunk.jep.*;
4  import java.awt.Rectangle;
5  import java.awt.image.BufferedImage;

7  public class Grafic extends javax.swing.JFrame {
8    int xDim=500;
9    int yDim=300;
```

```
11  public Grafic() {
12    initComponents();
13  }

15  private void initComponents() {
16    // cod generat de Netbeans
17  }

19        private void jButtonPlotMouseClicked(java.awt.event.MouseEvent evt){
20          JEP parser=new JEP();
21          parser.addStandardFunctions();
22          parser.addStandardConstants();
23          String var=jTextFieldVar.getText();
24          String fct=jTextFieldFct.getText();
25          String left=jTextFieldInf.getText();
26          parser.parseExpression(left);
27          double a=parser.getValue();
28          String right=jTextFieldSup.getText();
29          parser.parseExpression(right);
30          double b=parser.getValue();
31          parser.addVariable(var,0);
32          parser.parseExpression(fct);
33          Plot plotObj=new Plot();
34          int n=xDim*8;
35          double h,x,y;
36          if(b<a){
37            h=a;
38            a=b;
39            b=h;
40          }
41          h=(b-a)/n;
42          int dataset=0;
43          for(int i=0;i<=n;i++){
44            x=a+i*h;
45            parser.addVariable(var,x);
46            y=parser.getValue();
47            if((Double.isInfinite(y))||(Double.isNaN(y))){
48              dataset++;
49            }
50            else{
51              plotObj.addPoint(dataset,x,y,true);
52            }
53          }
54          Rectangle rectangle=new Rectangle(xDim,yDim);
55          BufferedImage image=plotObj.exportImage(rectangle);
56          ShowImage si=new ShowImage(image,xDim,yDim);
57          si.show();
58        }

60  public static void main(String args[]) {
61    java.awt.EventQueue.invokeLater(new Runnable() {
62      public void run() {
63        new Grafic().setVisible(true);
64      }
65    });
66  }

68  private javax.swing.JButton jButtonPlot;
69  private javax.swing.JLabel jLabelFct;
70  private javax.swing.JLabel jLabelInf;
71  private javax.swing.JLabel jLabelSup;
72  private javax.swing.JLabel jLabelVar;
73  private javax.swing.JTextField jTextFieldFct;
74  private javax.swing.JTextField jTextFieldInf;
75  private javax.swing.JTextField jTextFieldStatus;
76  private javax.swing.JTextField jTextFieldSup;
```

```
77    private javax.swing.JTextField jTextFieldVar;
78 }
```

Graficul obţinut pentru funcţia $f(x) = x - \sqrt{x^2 - 1},\ x \in [-5, 5]$ este dat în Fig. 7.2.

Fig. 7.2: Rezultatul reprezentării grafice.

7.2 *jfreechart*

În vederea compilării este necesar ca variabila de sistem `classpath` să conţină referinţa la fişierul *jfreechart-*.*.*.jar,* dar pentru execuţie trebuie declarată şi referinţa la *jcommon-*.*.*.jar,* aflată de asemenea în distribuţia produsului *jfreechart.*

jfreechart este destinat programatorilor, permiţând generarea mai multor tipuri de grafice *2D.* Exemplificăm prin reprezentarea grafică a unei funcţii reale de variabilă reală.

Reprezentările grafice se obţin prin intermediul obiectelor de tip `JFreeChart.` Un asemenea obiect se instanţiază prin metode statice ale clasei `ChartFactory.`

Metoda statică prin care se crează o reprezentare 2D unind punctele consecutive prin segmente este

```
public static JFreeChart createXYLineChart(java.lang.String title,
            java.lang.String xAxisLabel,
            java.lang.String yAxisLabel,
            XYDataset dataset,
            PlotOrientation orientation,
            boolean legend,
            boolean tooltips,
            boolean urls)
```

unde

- *title* este numele reprezentării grafice;

- *xAxisLabel* şi *yAxisLabel* sunt etichetele axelor Ox, respectiv Oy;
- *dataset* conţine una sau mai multe obiecte de tip `XYSeries`. Un obiect de tip `XYSeries` înmagazinează un şir de puncte, fixate prin coordonatele carteziene $(x_i, y_i)_i$. Pentru fiecare asemenea obiect, se reprezintă punctele unite prin segmente de dreaptă;
- *legend* indicator al prezenţei unei legende.

Denumim această reprezentare prin *XYLineChart.*

Pentru a obţine obiectul *dataset,* ce implementează interfaţa `XYDataset` se poate proceda după cum urmează:

1. Se instanţiază un obiect de tip `XYSeries`

   `XYSeries` *serie*`=new XYSeries(`*id*`)`

   unde *id* este un *String,* reprodus în legendă, prin care se identifică graficul corespunzător seriei de puncte.

2. Se completează variabila *serie* cu coordonatele punctelor

```
for(int i=0;i<n;i++){
  double x=. . .
  double y=. . .
  serie.add(x,y);
}
```

3. Se instanţiază variabilă *dataset* prin

   `XYDataset` *dataset*`=new XYSeriesCollection(`*serie*`);`

   Cu metoda clasei `XYSeriesCollection`

   `void addSeries(XYSeries` *serie*`),`

   se pot adăuga alte serii de date.

Astfel prelucrarea este

```
XYSeries serie=new XYSeries(‘‘f(x)’’);
for(int i=0;i<n;i++){
  double x=. . .
  double y=. . .
  serie.add(x,y);
}
XYDataset dataset=new XYSeriesCollection(series);
JFreeChart chart=ChartFactory(title,’’x’’,’’y’’,dataset,true,false,false);
```

Odată obiectul de *chart* tip `JFreeChart` creat, putem:

- obţine un obiect de tip `java.awt.image.BufferedImage` care poate fi desenat într-un obiect de tip `java.awt.Canvas:`

  `BufferedImage` *image=chart*`.createBufferedImage(` *xDim, yDim*`);`

  xDim $\times$ *yDim* reprezentând rezoluţia imaginii.

  Din nou programul *ShowImage* afişează imaginea obţinută pe ecranul monitorului.

- salva imaginea în format PNG sau JPEG într-un fişier

  `ChartUtilities.saveChartAsPNG(new java.io.File(`*numeFişier*`),` *chart, xDim, yDim*`);`

Exemplul 7.2.1 *Program pentru reprezentarea grafică a unei funcţii.*

Datele necesare se introduc prin intermediul aceleiaşi interfeţe grafice (Fig. 7.1)

Dacă a, b corespund marginilor inferioară şi respectiv superioară a intervalului în care se cere reprezentarea funcţiei $f(x)$, atunci ideea reprezentării constă din construirea şirului de puncte de coordonate $(x_i, y_i)_{0\leq i\leq n}$ cu $x_i = a + i\frac{b-a}{n},\ y_i = f(x_i),\ n \in \mathbb{N}^*$ şi de reprezentarea lor print-un grafic *XYLineChart*.

Pentru exemplul introdus $f(x) = x - \sqrt{x^2 - 1}$, domeniul de definiţie este $(-\infty, -1] \cup [1, \infty)$ şi în consecinţă graficul este alcătuit din două componente.

Evaluarea funcţiei în punctele ce nu aparţin domeniului de definiţie produce una din constantele `Double.POSITIVE(NEGATIVE)_INFINITY` sau `Double.NaN,` valori care nu se vor înregistra în obiectul `XYSeries.`

În acest caz reprezentarea *XYLineChart* este neconvenabilă - punctele de coordonate (-1,-1) şi (1,1) ar fi unite printr-un segment. Pentru depăşirea acestui inconvenient definim o reprezentare *XYDiscontinuousChart* prin metoda

```
1  public JFreeChart createXYDiscontinuousChart(String title,
2                                                String xAxisLabel,
3                                                String yAxisLabel,
4                                                XYDataset dataset,
5                                                PlotOrientation orientation,
6                                                boolean legend,
7                                                boolean tooltips,
8                                                boolean urls) {
9      NumberAxis xAxis = new NumberAxis(xAxisLabel);
10     xAxis.setAutoRangeIncludesZero(false);
11     NumberAxis yAxis = new NumberAxis(yAxisLabel);
12     XYItemRenderer renderer =
13       new StandardXYItemRenderer(StandardXYItemRenderer.DISCONTINUOUS_LINES);
14     XYPlot plot = new XYPlot(dataset, xAxis, yAxis, renderer);
15     plot.setOrientation(orientation);
16     if (tooltips){
17       renderer.setBaseToolTipGenerator(new StandardXYToolTipGenerator());
18     }
19     if (urls){
20       renderer.setURLGenerator(new StandardXYURLGenerator());
21     }
22     JFreeChart chart=
23       new JFreeChart(title,new Font("SansSerif",Font.BOLD,18),plot,legend);
24     return chart;
25   }
```

Această reprezentare se utilizează doar dacă numărul componentelor este cel puţin 2.

Pentru exemplul dat, va rezulta imaginea din Fig. 7.3

Codul sursă al programului este:

```
1 package plot2d;
2 import org.jfree.chart.*;
3 import org.jfree.data.xy.*;
```

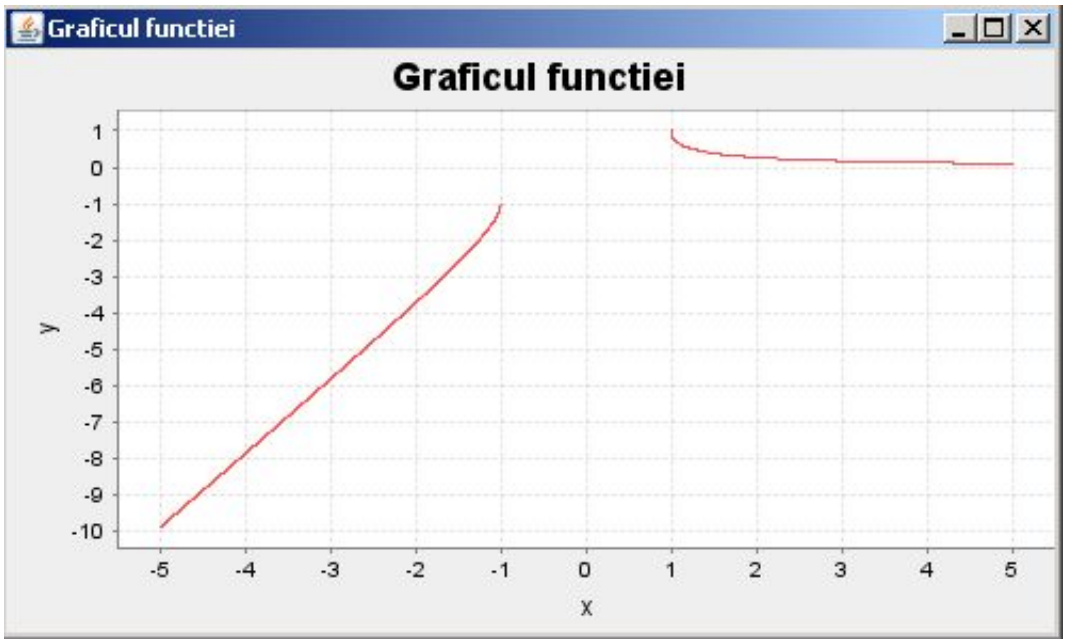


Fig. 7.3: Rezultatul reprezentării grafice.

```
4  import org.jfree.chart.renderer.xy.*;
5  import org.jfree.chart.plot.*;
6  import org.jfree.chart.axis.*;
7  import org.jfree.chart.labels.*;
8  import org.jfree.chart.urls.*;
9  import java.awt.*;
10 import java.awt.image.*;
11 import org.nfunk.jep.*;

13 public class Grafic extends javax.swing.JFrame {
14   int xDim=500;
15   int yDim=300;

17   public Grafic() {
18     initComponents();
19   }

21   private void initComponents() {
22     //cod generat de Netbeans corespunzatoare interfatei grafice
23   }

25         private void jButtonPlotMouseClicked(java.awt.event.MouseEvent evt){
26           JEP parser=new JEP();
27           parser.addStandardFunctions();
28           parser.addStandardConstants();
29           String var=jTextFieldVar.getText();
30           String fct=jTextFieldFct.getText();
31           String left=jTextFieldInf.getText();
32           parser.parseExpression(left);
33           double a=parser.getValue();
34           String right=jTextFieldSup.getText();
35           parser.parseExpression(right);
36           double b=parser.getValue();
37           parser.addVariable(var,0);
38           parser.parseExpression(fct);
39           XYSeries series = new XYSeries("y");
40           int n=xDim*8;
```

```
41            byte[] s=new byte[n+1];
42            double h,x,y;
43            if(b<a){
44              h=a;
45              a=b;
46              b=h;
47            }
48            h=(b-a)/n;
49            for(int i=0;i<=n;i++){
50              x=a+i*h;
51              parser.addVariable(var,x);
52              y=parser.getValue();
53              if((Double.isInfinite(y))||(Double.isNaN(y))){
54                s[i]=1;
55              }
56              else{
57                series.add(x, y);
58                s[i]=0;
59              }
60            }
61            XYDataset dataset = new XYSeriesCollection(series);
62            JFreeChart chart=null;
63              if(isCompact(s)){
64                chart = ChartFactory.createXYLineChart(
65                        "Graficul functiei","x","y",
66                    dataset,org.jfree.chart.plot.PlotOrientation.VERTICAL,
67                    true,false,false);
68              }
69              else{
70                chart = createXYDiscontinuousChart(
71                    "Graficul functiei","x","y",
72                    dataset,org.jfree.chart.plot.PlotOrientation.VERTICAL,
73                    true,false,false);
74              }
75              BufferedImage image=chart.createBufferedImage(xDim,yDim);
76              ShowImage si=new ShowImage(image,xDim,yDim);
77              si.show();
78              try {
79                ChartUtilities.saveChartAsPNG(new java.io.File("Functia.png"),
80                  chart, xDim, yDim);
81              }
82              catch (java.io.IOException e){
83                System.err.println("Error writing image to file : "+
84                  e.getMessage());
85              }
86          }

88    public JFreeChart createXYDiscontinuousChart(. . .){. . .}

90    boolean isCompact(byte[] s){
91      int n=s.length;
92      int noNaN=0,no=0;
93      boolean first=true;
94      for(int i=0;i<n;i++){
95       if((s[i]==0)&& first){
96         first=false;
97         no++;
98         }
99       if((s[i]==1)&& (!first)) noNaN++;
100        if((s[i]==0)&& (!first) && (noNaN>=1)) {
101            no++;
102            break;
103        }
104     }
105     if(no>1)
106         return false;
107     else
```

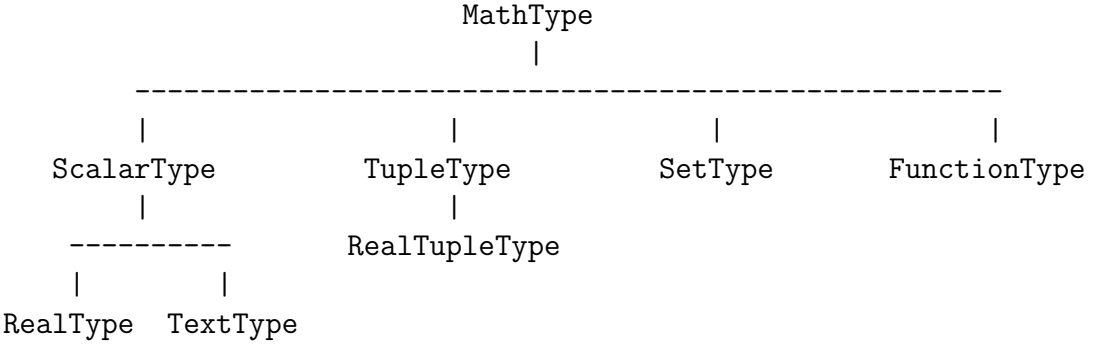
```
108         return true;
109   }

111   public static void main(String args[]) {
112     java.awt.EventQueue.invokeLater(new Runnable() {
113       public void run() {
114         new Grafic().setVisible(true);
115       }
116     });
117   }

119   private javax.swing.JButton jButtonPlot;
120   private javax.swing.JLabel jLabelFct;
121   private javax.swing.JLabel jLabelInf;
122   private javax.swing.JLabel jLabelSup;
123   private javax.swing.JLabel jLabelVar;
124   private javax.swing.JTextField jTextFieldFct;
125   private javax.swing.JTextField jTextFieldInf;
126   private javax.swing.JTextField jTextFieldStatus;
127   private javax.swing.JTextField jTextFieldSup;
128   private javax.swing.JTextField jTextFieldVar;
129 }
```

7.3 *VisAD*

VisAD – Visualization for Algorithmic Development este un pachet Java, care permite obţinerea de reprezentări grafice în două (2D) şi trei dimensiuni (3D), de animaţii, `ftp://ftp.ssec.wisc.edu/pub/visad-2.0/visad.jar`. La dezvoltarea pachetului au contribuit mai multe universităţi şi instituţii, începând din 1992. *VisAD* se bazează pe un model de reprezentare prin obiecte a diverselor entităţi ce apar în construirea unei reprezentări grafice. Fixarea datelor de reprezentat grafic, care apar uzual într-o formulare matematică, în obiectele modelului este relativ simplă şi cade în sarcina programatorului, la fel şi fixarea unor parametri privind modul de afişare. Toate operaţiile care conduc la imaginea grafică sunt complet transparente programatorului.

Utilizarea produsului necesită în plus softul *java3d* de la *Oracle.*

Ierarhia de clase Java care fixează structurile de date utilizabile în *VisAD* este



Dintre aceste clase precizăm:

- `RealType` serveşte la declararea unei variabile numerice.

  Instanţierea unui obiect se programează prin metoda statică

`RealType getRealType(String` *numeVar*`)`

- `RealTupleType` serveşte la declararea de variabile din $\mathbb{R}^n$. Constructori:

  `RealTupleType(RealType[])`

  `RealTupleType(RealType,RealType)`

- `FunctionType` serveşte la declararea funcţiilor, având constructorul

  `FunctionType(MathType` *domDef* `,MathType` *mulţimeVal*`)`

În următoarele două exemple introducem şi alte clase ale pachetului *VisAD* care intervin la construirea unei reprezentări grafice şi arătăm un şablon de lucru.

Reprezentarea grafică a unei funcţii de o variabilă

Construirea reprezentării grafice a unei funcţii $fct : [a, b] \cap D \to \mathbb{R}$ se obţine parcurgând paşii:

1. Definirea structurilor de date

   ```
   RealType xType = RealType.getRealType("x");
   RealType yType = RealType.getRealType("y");
   FunctionType fctType = new FunctionType(xType, yType);
   ```

2. Precizarea datelor: în intervalul $[a, b]$ se defineşte o reţea echidistantă de $n \in \mathbb{N}^*$ puncte

   ```
   Linear1DSet xSet = new Linear1DSet(xType, a, b, n);
   ```

   Tabloul absciselor este generat prin

   ```
   float[][] xValues=xSet.getSamples(true);
   ```

   iar mulţimea valorilor funcţiei se calculează prin

   ```
   double[][] yValues = new double[1][n];
   for(int i=0;i<n;i++){
     yValues[0][i]=fct(xValues[0][i]);
   }
   ```

3. Fixarea conexiunii dintre structurile de date şi tablourile de valori

   ```
   FlatField ff = new FlatField( fctType, xSet);
   ff.setSamples( yValues );
   ```

4. Fixarea elementelor care definesc reprezentarea grafică

- Structurile de date:

```
ScalarMap xMap = new ScalarMap( xType, Display.XAxis );
ScalarMap yMap = new ScalarMap( yType, Display.YAxis );
```

- Conexiunea dintre structurile de date şi tablourile de valori

```
DataReferenceImpl dri=new DataReferenceImpl("data_ref");
dri.setData(ff);
```

5. Definirea unui container grafic căruia i se adaugă elementele reprezentării grafice

```
DisplayImpl display = new DisplayImplJ2D("2d");
display.addMap( xMap );
display.addMap( yMap );
display.addReference(dri);
```

6. Opţional, reprezintarea axelor se programează prin

```
GraphicsModeControl
  gmc=(GraphicsModeControl)display.getGraphicsModeControl();
gmc.setScaleEnable(true);
```

7. Containerul grafic se integrează într-un cadru *swing*

```
JFrame jframe = new JFrame();
jframe.getContentPane().add(display.getComponent(),
  BorderLayout.CENTER);
```

Exemplul 7.3.1 *Să se reprezinte graficul funcţiei* $fct(x) = x - \sqrt{x^2 - 1},\ x \in [-5, 5] \setminus (-1, 1)$.

Pentru simplitate, datele problemei sunt fixate în clasa *DataIn*

```
1  package grafic2d;
2  public class DataIn{
3    private double a=-5;
4    private double b=5;
5    private int n=100;

7    public double fct(double x){
8      return x-Math.sqrt(x*x-1);
9    }

11   public double getA(){
12     return a;
13   }

15   public double getB(){
16     return b;
17   }

19   public int getN(){
20     return n;
21   }
22 }
```

Reprezentarea grafică se generează în clasa

```
1  package grafic2d;
2  import visad.*;
3  import visad.java2d.DisplayImplJ2D;
4  import java.rmi.RemoteException;
5  import java.awt.*;
6  import javax.swing.*;

8  public class ReprezentareFunctie2D{
9    private DataIn din;

11   public ReprezentareFunctie2D(DataIn din){
12     this.din=din;
13   }

15   public void plot(){
16     int n=din.getN();
17     DisplayImpl display=null;

19     try{
20       // Definirea structurilor de date
21       RealType xType = RealType.getRealType("x");
22       RealType yType = RealType.getRealType("y");
23       FunctionType fctType = new FunctionType(xType, yType);
24       // Precizarea datelor
25       Linear1DSet xSet = new Linear1DSet(xType,din.getA(),din.getB(),n);
26       float[][] xValues=xSet.getSamples(true);
27       double[][] yValues = new double[1][n];
28       for(int i=0;i<n;i++){
29         yValues[0][i]=din.fct(xValues[0][i]);
30       }
31       // Fixarea conexiunii dintre structurile de date
32       // si tablourile de valori
33       FlatField ff = new FlatField( fctType, xSet);
34       ff.setSamples( yValues );

36       // Fixarea elementelor care definesc reprezentare grafica
37       ScalarMap xMap = new ScalarMap( xType, Display.XAxis );
38       ScalarMap yMap = new ScalarMap( yType, Display.YAxis );
39       DataReferenceImpl dri=new DataReferenceImpl("data_ref");
40       dri.setData(ff);
41       // Definirea containerului grafice caruia i
42       // se adauga elementele reprezentarii grafice
43       display = new DisplayImplJ2D("2d");
44       display.addMap( xMap );
45       display.addMap( yMap );
46       display.addReference(dri);
47       // Desenarea axelor de coordonate
48       GraphicsModeControl gmc=
49         (GraphicsModeControl)display.getGraphicsModeControl();
50       gmc.setScaleEnable(true);
51     }
52     catch(VisADException e){
53       System.out.println("VisadException : "+e.getMessage());
54       System.exit(1);
55     }
56     catch(RemoteException e){
57       System.out.println("RMI-RemoteException : "+e.getMessage());
58       System.exit(1);
59     }
60     // Integrarea intr-un cadrul swing
61     JFrame jframe = new JFrame("Graficul functiei");
62     jframe.getContentPane().setLayout(new BorderLayout());
63     jframe.getContentPane().add(display.getComponent(),BorderLayout.CENTER);
64     jframe.setDefaultCloseOperation(jframe.EXIT_ON_CLOSE);
65     jframe.setSize(400, 400);
```

```
66      jframe.setVisible(true);
67    }

69    public static void main(String[] args){
70      DataIn din=new DataIn();
71      ReprezentareFunctie2D obj=new ReprezentareFunctie2D(din);
72      obj.plot();
73    }
74  }
```

Va rezulta imaginea din Fig. 7.4

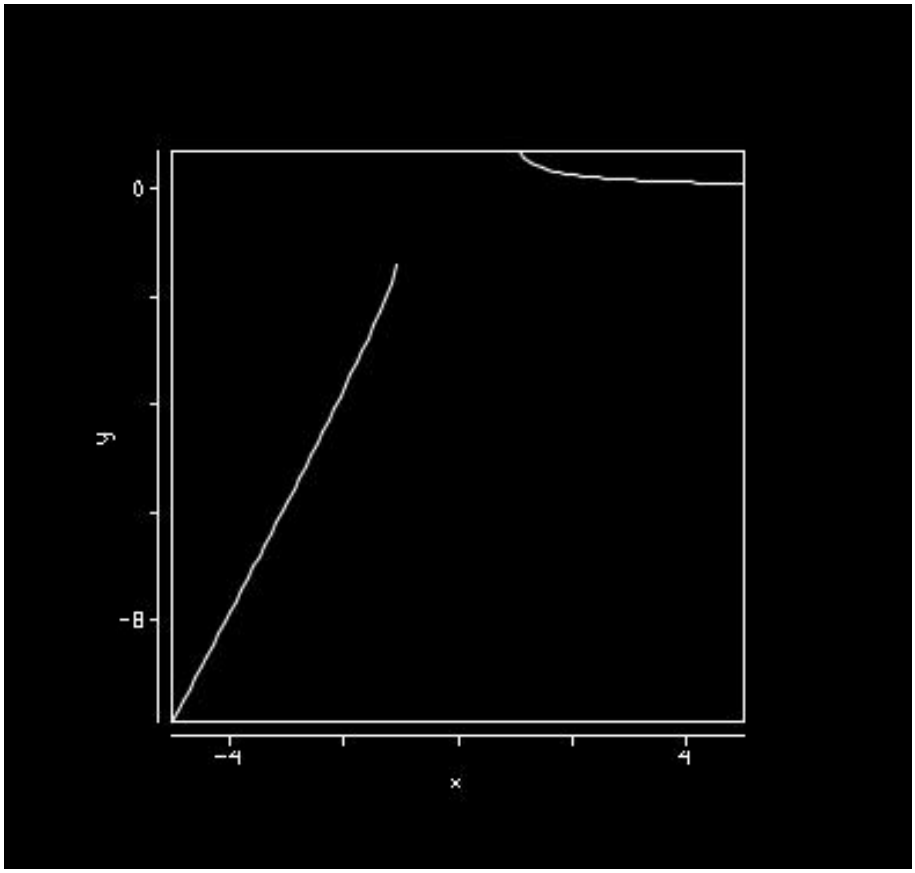


Fig. 7.4: Graficul 2D produs de *VisAD*.

Reprezentarea grafică a unei funcţii de două variabile

Procedăm asemănător pentru construirea reprezentării grafice a unei funcţii $fct : ([x_{Min}, x_{Max}] \times [y_{Min}, y_{Max}]) \cap D \to \mathbb{R}$:

1. Definirea structurilor de date

```
RealType yType = RealType.getRealType("y");
```

```
RealType zType = RealType.getRealType("z");
RealTupleType xyType = new RealTupleType(xType,yType);
FunctionType fctType = new FunctionType(xyType, zType);
```

2. Precizarea datelor: în dreptunghiul $[x_{Min}, x_{Max}] \times [y_{Min}, y_{Max}]$ se defineşte o reţea bidimensională de puncte echidistante, în număr de $nx \times ny$

   ```
   Linear2DSet xySet =
      new Linear2DSet(xyType,XMin(),XMax(),nx,YMin(),YMax(),ny);
   ```

   Tabloul punctelor este generat prin

   ```
   float[][] xyValues=xySet.getSamples(true);
   ```

   iar mulţimea valorilor funcţiei se calculează prin

   ```
   double[][] zValues = new double[1][nx*ny];
   for(int i=0;i<nx*ny;i++){
     zValues[0][i]=din.fct(xyValues[0][i],xyValues[1][i]);
   }
   ```

3. Fixarea conexiunii dintre structurile de date şi tablourile de valori

   ```
   FlatField ff = new FlatField(fctType, xySet);
   ff.setSamples( zValues);
   ```

4. Fixarea elementelor care definesc reprezentarea grafică

   - Structurile de date:

     ```
     ScalarMap xMap = new ScalarMap(xType, Display.XAxis );
     ScalarMap yMap = new ScalarMap(yType, Display.YAxis );
     ScalarMap zMap = new ScalarMap(zType, Display.ZAxis );
     ScalarMap zColMap = new ScalarMap(zType, Display.RGB);
     ```

     Ultimul rând are ca efect afişarea în culori a suprafeţei, punctele cu valori mici ale lui z se reprezintă în albastru iar cele cu valori mari în roşu.

   - Conexiunea dintre structurile de date şi tablourile de valori

     ```
     DataReferenceImpl dri=new DataReferenceImpl("data_ref");
     dri.setData(ff);
     ```

5. Definirea unui container grafic căruia i se adaugă elementele reprezentării grafice

```
DisplayImpl display = new DisplayImplJ3D("3d");
display.addMap( xMap );
display.addMap( yMap );
display.addMap( zMap );
display.addMap( zColMap );
display.addReference(dri);
```

6. Opţional, reprezentarea axelor se programează prin

```
GraphicsModeControl
  gmc=(GraphicsModeControl)display.getGraphicsModeControl();
gmc.setScaleEnable(true);
gmc.setTextureEnable(false);
```

7. Containerul grafic se integrează într-un cadru *swing*

```
JFrame jframe = new JFrame();
jframe.getContentPane().add(display.getComponent(),
  BorderLayout.CENTER);
```

Exemplul 7.3.2 *Să se reprezinte graficul funcţiei* $fct(x,y) = x*x-y*y,\ x \in [-5,5], y \in [-5,5]$.

Din nou se fixează datele problemei în clasa *DataIn*

```
1  package grafic3d;
2  public class DataIn{
3    private double xMin=-5;
4    private double xMax=5;
5    private double yMin=-5;
6    private double yMax=5;
7    private int nx=30;
8    private int ny=30;

10   public double fct(double x,double y){
11     return x*x-y*y;
12   }

14   public double getXMin(){
15     return xMin;
16   }

18   public double getXMax(){
19     return xMax;
20   }

22   public double getYMin(){
23     return yMin;
24   }

26   public double getYMax(){
27     return yMax;
28   }

30   public int getNx(){
31     return nx;
```

```
32    }

34    public int getNy(){
35      return ny;
36    }
37  }
```

Programul reprezentării grafice are codul

```
1  package grafic3d;
2  import visad.*;
3  import visad.java3d.DisplayImplJ3D;
4  import java.rmi.RemoteException;
5  import java.awt.*;
6  import javax.swing.*;

8  public class ReprezentareFunctie3D{
9    private DataIn din;

11   public ReprezentareFunctie3D(DataIn din){
12     this.din=din;
13   }

15   public void plot3D(){
16     int nx=din.getNx();
17     int ny=din.getNy();
18     DisplayImpl display=null;

20     try{
21       // Definirea structurii de date
22       RealType xType = RealType.getRealType("x");
23       RealType yType = RealType.getRealType("y");
24       RealType zType = RealType.getRealType("z");
25       RealTupleType xyType = new RealTupleType(xType,yType);
26       FunctionType fctType = new FunctionType(xyType, zType);

28       // Precizarea datelor
29       Linear2DSet xySet=new Linear2DSet(xyType,din.getXMin(),din.getXMax(),
30         nx,din.getYMin(),din.getYMax(),ny);
31       float[][] xyValues=xySet.getSamples(true);
32       double[][] zValues = new double[1][nx*ny];
33       for(int i=0;i<nx*ny;i++){
34         zValues[0][i]=din.fct(xyValues[0][i],xyValues[1][i]);
35       }

37       // Fixarea conexiunii dintre structurile de date
38       // si tablourile de valori
39       FlatField ff = new FlatField( fctType, xySet);
40       ff.setSamples( zValues);

42       // Fixarea elementelor care definesc reprezentare grafica
43       ScalarMap xMap = new ScalarMap(xType, Display.XAxis );
44       ScalarMap yMap = new ScalarMap(yType, Display.YAxis );
45       ScalarMap zMap = new ScalarMap(zType, Display.ZAxis );
46       ScalarMap zColMap = new ScalarMap(zType, Display.RGB);
47       DataReferenceImpl dri=new DataReferenceImpl("data_ref");
48       dri.setData(ff);

50       // Definirea containerului grafice caruia i
51       // se adauga elementele reprezentarii grafice
52       display = new DisplayImplJ3D("3d");
53       display.addMap( xMap );
54       display.addMap( yMap );
55       display.addMap( zMap );
56       display.addMap( zColMap );
57       display.addReference(dri);
```

```
59        // Elemente suplimentare de control
60        GraphicsModeControl gmc=
61          (GraphicsModeControl)display.getGraphicsModeControl();
62        gmc.setScaleEnable(true);
63        gmc.setTextureEnable(false);
64      }
65      catch(VisADException e){
66        System.out.println("VisadException : "+e.getMessage());
67        System.exit(1);
68      }
69      catch(RemoteException e){
70        System.out.println("RMI-RemoteException : "+e.getMessage());
71        System.exit(1);
72      }

74      // Integrarea intr-un cadru swing
75      JFrame jframe=new JFrame("Graficul 3D al unei functii in 2 variabile");
76      jframe.getContentPane().setLayout(new BorderLayout());
77      jframe.getContentPane().add(display.getComponent(),BorderLayout.CENTER);
78      jframe.setDefaultCloseOperation(jframe.EXIT_ON_CLOSE);
79      jframe.setSize(600, 600);
80      jframe.setVisible(true);
81    }
82    public static void main(String[] args){
83      DataIn din=new DataIn();
84      ReprezentareFunctie3D obj=new ReprezentareFunctie3D(din);
85      obj.plot3D();
86    }
87  }
```

Va rezulta imaginea din Fig. 7.5.

Imaginea reprodusă este după rotirea cu ajutorul mouse-ului a imaginii generată iniţial.

7.4 Vizualizarea funcţiilor complexe

Ne propunem vizualizarea funcţiei compleze $f : D \subset \mathbb{C} \to \mathbb{C}$ unde $D = \{z = x + iy : x \in [x_m, x_M], y \in [y_m, y_M]\}$. Ideea vizualizării, potrivit [8], constă în atribuirea unei culori fiecărui număr complex. Pentru un punct $z \in D$ se calculează $f(z)$, acestuia îi corespunde o culoare $\mathcal{C}$, iar vizualizarea funcţiei se obţine afişând punctul z în culoarea $\mathcal{C}$.

O imagine grafică se va obţine într-o fereastra a ecranului

$$\mathbb{D} = \{0, 1, \ldots, P\} \times \{0, 1, \ldots, Q\}, \qquad P, Q \in \mathbb{N}.$$

Funcţia bijectivă $\varphi : \mathbb{D} \to D$ definită prin

$$\varphi(p, q) = \left(x_m + \frac{x_M - x_m}{P} p\right) + i \left(y_M - \frac{y_M - y_m}{Q} q\right)$$

atribuie fiecărui pixel $(p, q) \in \mathbb{D}$ un număr complex din D, Fig. 7.6. Culoarea $\mathcal{C}$ corespunzătoare numărului complex $f(\varphi(p, q))$ va fi atribuită pixelului de coordonate (p, q), $p \in \{0, 1, \ldots, P\}$, $p \in \{0, 1, \ldots, Q\}$. În felul acesta se obţine vizualizarea funcţiei complexe f.

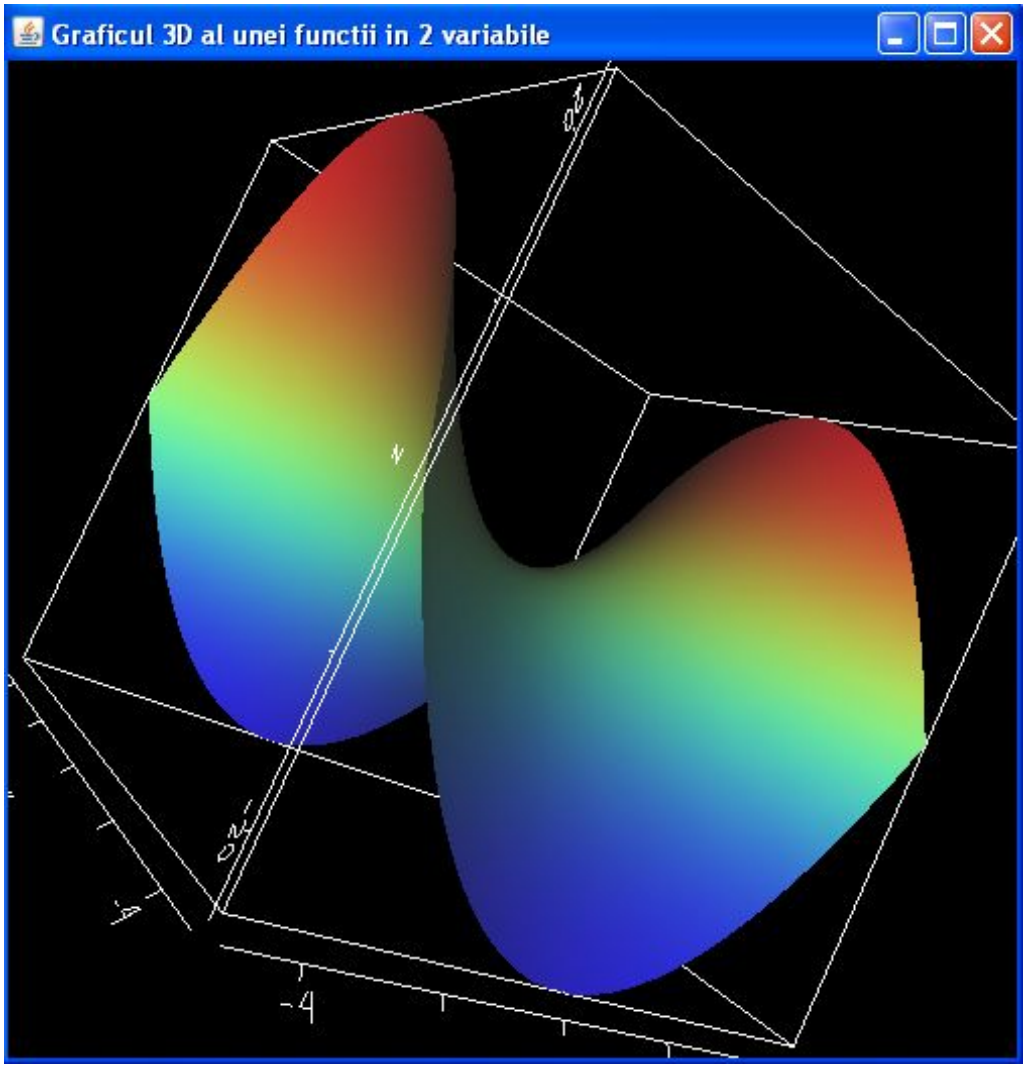


Fig. 7.5: Graficul 3D produs de *VisAD*.

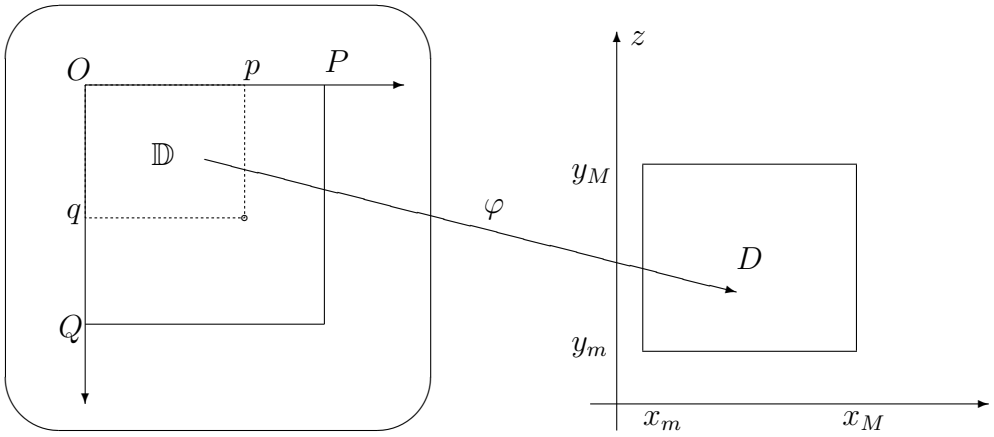


Fig. 7.6: Transformarea $\varphi : \mathbb{D} \to D$.

Cubul culorilor

O culoare se formează dintr-un triplet $(r, g, b) \in [0, 1]^3$. Interpretăm mulţimea $[0, 1]^3$ ca un cub ale cărui vârfuri definesc culorile potrivit Fig. 7.7.

În Java, atribuirea unei culori (r, g, b) unui pixel (p, q) se programează prin

```
graphics.setColor(new Color(r,g,b));
graphics.fillRect(p,q,1,1);
```

unde *graphics* este un obiect de tip `java.awt.Graphics` ataşat mediului (virtual) de desenare, de exemplu

```
BufferedImage bi=new BufferedImage(P,Q,BufferedImage.TYPE_3BYTE_BGR);
Graphics graphics=bi.getGraphics();
```

Proiecţia stereografică

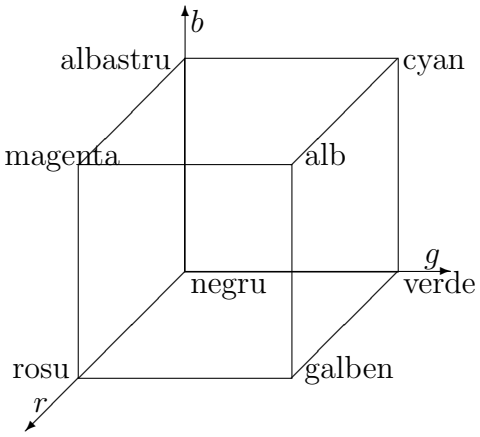
Proiecţia stereografică realizează o bijecţie între planul complex completat cu elementul ∞ şi suprafaţa S a sferei cu centrul în origine şi de rază 1

$$S: \qquad X^2 + Y^2 + Z^2 = 1.$$

Numărului complex $z = x + iy$ i se asociază punctul N, intersecţia dreptei definită de punctele $M(x, y, 0)$ şi $A(0, 0, 1)$ cu sfera S, Fig. 7.8.

Ecuaţiile dreptei AM sunt $\frac{X}{x} = \frac{Y}{y} = \frac{Z-1}{-1}$. Intersecţia dreptei AM cu sfera S are coordonatele

$$X_N = \frac{2x}{x^2 + y^2 + 1}, \qquad Y_N = \frac{2y}{x^2 + y^2 + 1}, \qquad Z_N = 1 - \frac{2}{x^2 + y^2 + 1}.$$



(r, g, b)	Culoarea
(0,0,0)	negru
(1,0,0)	roşu
(0,1,0)	verde
(0,0,1)	albastru
(1,1,0)	galben
(1,0,1)	magenta
(0,1,1)	cyan
(1,1,1)	alb

Fig. 7.7: Cubul culorilor.

Dacă $|z| = \sqrt{x^2 + y^2} > 1$ atunci intersecţia se află pe semisfera superioară, iar dacă $|z| < 1$ atunci intersecţia se află pe semisfera inferioară. Elementului ∞ îi corespunde punctul A, iar lui 0 îi corespunde punctul diametral opus lui A.

Ideea atribuirii unei culori numărului complex z, adică a transformării coordonatelor proiecţiei stereografice într-un punct al cubului $[0, 1]^3$ constă din următoarele operaţii:

1. O omotetie de centru O de factor $\frac{1}{2}$, care transformă sfera S într-o sferă de rază $\frac{1}{2}$;

2. O rotaţie ce duce axa OZ pe direcţia *negru,alb;*

3. O translaţie ce duce centrul sferei în punctul de coordonate $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.

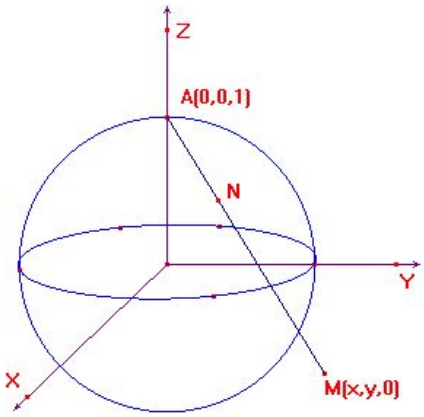
Procedând în acest fel, culorile alb şi negru nu sunt atribuite nici unui punct, mai mult dispersia culorilor este redusă. Pentru a remedia acest lucru, modificăm valoarea lui Z. Mai precis, Z va fi cota corespunzătoare punctului

$$\left(\frac{2x}{x^2 + y^2 + 1}, \frac{2x}{x^2 + y^2 + 1}, 0\right)$$

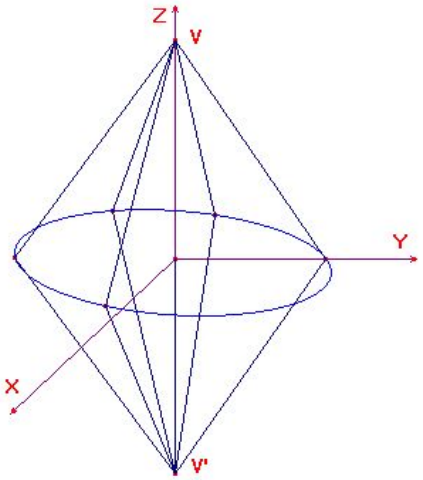
pe conul având ca baza cercul $X^2+Y^2 = 1, Z = 0$ şi vârful $V(0, 0, \sqrt{3})$ - în cazul punctului de pe semisfera superioară, respectiv, conul cu aceaşi bază dar cu vârful $V'(0, 0, -\sqrt{3})$ - în cazul punctului de pe semisfera inferioară.

O dreaptă care trece prin V

$$\begin{cases} X = \lambda(Z - \sqrt{3}) \\ Y = \mu(Z - \sqrt{3}) \end{cases} \tag{7.1}$$



Proiecţia stereografică



Construcţia
pentru modificarea lui Z.

Fig. 7.8:

este generatoare a conului dacă intersectează cercul bazei. Rezultă condiţia de compatibilitate

$$3(\lambda^2 + \mu^2) = 1. \tag{7.2}$$

Eliminând parametrii λ şi μ între relaţiile (7.1) şi (7.2) se obţine ecuaţia conului superior

$$Z = \sqrt{3}(1 - \sqrt{X^2 + Y^2}).$$

Ecuaţia conului inferior se obţine analog

$$Z = \sqrt{3}(\sqrt{X^2 + Y^2} - 1).$$

Prin urmare

$$Z = \sqrt{3}\ \mathrm{sgn}(x^2 + y^2 - 1)\left(1 - 2\frac{\sqrt{x^2 + y^2}}{x^2 + y^2 + 1}\right).$$

Fie N' punctul de coordonate

$$\begin{aligned} X_{N'} &= X_N = \tfrac{2x}{x^2+y^2+1} \\ Y_{N'} &= Y_N = \tfrac{2y}{x^2+y^2+1} \\ Z_{N'} &= \sqrt{3}\ \mathrm{sgn}(x^2 + y^2 - 1)\left(1 - 2\tfrac{\sqrt{x^2+y^2}}{x^2+y^2+1}\right) \end{aligned}$$

Efectuăm cele trei operaţii descrise anterior care duc corpul format de cele două conuri (7.8) în cubul culorilor $[0, 1]^3$.

1. Transformarea de omotetie a punctului N' are ca rezultat punctul N_1 de coordonate

$$\begin{aligned} X_{N_1} &= \frac{x}{x^2+y^2+1} \\ Y_{N_1} &= \frac{y}{x^2+y^2+1} \\ Z_{N_1} &= \sqrt{3}\ \mathrm{sgn}(x^2+y^2-1)\left(\frac{1}{2}-\frac{\sqrt{x^2+y^2}}{x^2+y^2+1}\right) \end{aligned}$$

2. Rotaţia. Notând prin $\vec{I},\vec{J},\vec{K}$ versorii axelor de coordonate în sistemul XYZ şi cu $\vec{\imath},\vec{\jmath},\vec{k}$ versorii axelor în spaţiul cubului culorilor rgb, avem (Fig. 7.9)

$$\vec{Z}=\frac{1}{\sqrt{3}}(\vec{\imath}+\vec{\jmath}+\vec{k}).$$

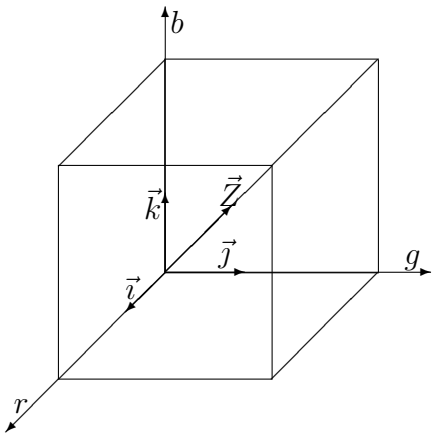


Fig. 7.9: Versori în cubul culorilor.

Versorii $\vec{I}$ şi $\vec{J}$ sunt perpendiculari pe $\vec{K}$, deci admit reprezentări de forma

$$\begin{aligned} \vec{I} &= a\vec{\imath}+b\vec{\jmath}-(a+b)\vec{k}, \quad a,b\in\mathbb{R}, a\geq 0, \\ \vec{J} &= c\vec{\imath}+d\vec{\jmath}-(c+d)\vec{k}, \quad c,d\in\mathbb{R}, c\geq 0. \end{aligned}$$

Condiţia $|\vec{I}|=1$ implică $a^2+ab+b^2=\frac{1}{2}$ de unde $b=\frac{-a\pm\sqrt{2-3a^2}}{2}$.

Analog, rezultă $d=\frac{-c\pm\sqrt{2-3c^2}}{2}$.

Prin urmare

$$\begin{aligned} \vec{I} &= a\vec{\imath}+\tfrac{-a\pm\sqrt{2-3a^2}}{2}\vec{\jmath}-\tfrac{a\pm\sqrt{2-3a^2}}{2}\vec{k}, \\ \vec{J} &= c\vec{\imath}+\tfrac{-c\pm\sqrt{2-3c^2}}{2}\vec{\jmath}-\tfrac{c\pm\sqrt{2-3c^2}}{2}\vec{k}. \end{aligned}$$

Condiţia de ortogonalitate $\vec{I}\perp\vec{J} \Leftrightarrow \vec{I}\cdot\vec{J}=0$ implică

$$ac+\frac{-a\pm\sqrt{2-3a^2}}{2}\cdot\frac{-c\pm\sqrt{2-3c^2}}{2}+\frac{a\pm\sqrt{2-3a^2}}{2}\cdot\frac{c\pm\sqrt{2-3c^2}}{2}=0. \quad (7.3)$$

Dacă pentru $\pm$ alegem simultan acelaşi semn atunci relaţia anterioară devine $3ac + \sqrt{(2-3a^2)(2-3c^2)} = 0$, relaţie care nu poate avea loc.

Prin urmare semnele $\pm$ în $\vec{I}$ şi $\vec{J}$ trebuie să fie contrare, de unde

$$\begin{aligned}\vec{I} &= a\vec{\imath} + \tfrac{-a+\sqrt{2-3a^2}}{2}\vec{\jmath} - \tfrac{a+\sqrt{2-3a^2}}{2}\vec{k},\\ \vec{J} &= c\vec{\imath} + \tfrac{-c-\sqrt{2-3c^2}}{2}\vec{\jmath} - \tfrac{c-\sqrt{2-3c^2}}{2}\vec{k}.\end{aligned}$$

Condiţia de ortogonalitate (7.3) devine $3ac - \sqrt{(2-3a^2)(2-3c^2)} = 0$, de unde se obţine

$$c = \sqrt{\frac{2-3a^2}{3}}.$$

Valoarea lui a se determină astfel încât axa OX să fie cât mai apropiată de axa Or din spaţiul culorilor. În acest fel, valorile reale se vor reprezenta în roşu.

Exprimăm condiţia de apropiere prin cerinţa ca expresia

$$|\vec{I} - \vec{\imath}| \tag{7.4}$$

să fie minimă.

Prin calcul direct $|\vec{I} - \vec{\imath}|^2 = 2 - 2a$. Condiţiile $2 - 3a^2 \geq 0, a \geq 0$ implică $a \in [0, \frac{2}{\sqrt{6}}]$. Minimul expresiei (7.4) se obţine pentru $a = \frac{2}{\sqrt{6}}$. În consecinţă

$$\begin{aligned}\vec{I} &= \tfrac{2}{\sqrt{6}}\vec{\imath} - \tfrac{1}{\sqrt{6}}\vec{\jmath} - \tfrac{2}{\sqrt{6}}\vec{k},\\ \vec{J} &= \tfrac{1}{\sqrt{2}}\vec{\imath} - \tfrac{1}{\sqrt{2}}\vec{\jmath},\end{aligned}$$

sau

$$\begin{bmatrix}\vec{I}\\ \vec{J}\\ \vec{K}\end{bmatrix} = \begin{pmatrix}\frac{2}{\sqrt{6}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{6}}\\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}}\\ \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}}\end{pmatrix} \cdot \begin{bmatrix}\vec{\imath}\\ \vec{\jmath}\\ \vec{k}\end{bmatrix}.$$

În urma rotaţiei punctul N_1 se transformă într-un punct N_2 ale cărui coordonate sunt

$$\begin{pmatrix}x_{N_2}\\ y_{N_2}\\ z_{N_2}\end{pmatrix} = \begin{pmatrix}\frac{2}{\sqrt{6}} & 0 & \frac{1}{\sqrt{3}}\\ -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}}\\ -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}}\end{pmatrix} \cdot \begin{pmatrix}x_{N_1}\\ y_{N_1}\\ z_{N_1}\end{pmatrix}.$$

3. În urma translaţiei din spaţiul cubului culorilor, formulele de calcul ale componentelor culorii devin

$$\begin{aligned}r &= \tfrac{1}{2} + x_{N_2} = \tfrac{1}{2} + \tfrac{2}{\sqrt{6}}\tfrac{x}{x^2+y^2+1} + \operatorname{sgn}(\mathrm{x}^2+\mathrm{y}^2-1)(\tfrac{1}{2} - \tfrac{\sqrt{x^2+y^2}}{x^2+y^2+1})\\ g &= \tfrac{1}{2} + y_{N_2} = \tfrac{1}{2} - \tfrac{1}{\sqrt{6}}\tfrac{x}{x^2+y^2+1} + \tfrac{1}{\sqrt{2}}\tfrac{y}{x^2+y^2+1} + \operatorname{sgn}(\mathrm{x}^2+\mathrm{y}^2-1)(\tfrac{1}{2} - \tfrac{\sqrt{x^2+y^2}}{x^2+y^2+1})\\ b &= \tfrac{1}{2} + z_{N_2} = \tfrac{1}{2} - \tfrac{1}{\sqrt{6}}\tfrac{x}{x^2+y^2+1} - \tfrac{1}{\sqrt{2}}\tfrac{y}{x^2+y^2+1} + \operatorname{sgn}(\mathrm{x}^2+\mathrm{y}^2-1)(\tfrac{1}{2} - \tfrac{\sqrt{x^2+y^2}}{x^2+y^2+1})\end{aligned} \tag{7.5}$$

Aplicaţia de vizualizare

Formulele (7.5) le vom folosi într-o aplicaţie de vizualizarea unei funcţii complexe. Pentru orice funcţie f, se vor afişa două panouri, unul corespunzând vizualizării funcţiei $z \longmapsto z$, cu rol de calibrare a percepţiei vizuale, iar celălalt panou destinat vizualizării funcţiei f, $z \in D$. Cele două reprezentări sunt programate în metodele *refPlot* şi *fctPlot* ale clasei *ComplexPlot.*

În plus, dacă $x_m = -2, x_M = 2, y_m = -2, y_M = 2$ atunci sunt reprezentate dreptele $\Im z = 0, \Re z = 0, \arg(z) = \pm\frac{\pi}{4}$ şi cercurile $|z| = \frac{1}{2}, |z| = 1, |z| = \frac{3}{2}$, respectiv imaginile lor prin f.

Clasele Java ale aplicaţiei au codurile

```
1  package complexplot;
2  import java.awt.image.BufferedImage;
3  import java.awt.Graphics;
4  import java.awt.Color;
5  import org.nfunk.jep.type.Complex;

7  public class ComplexPlot {
8    DataIn din=null;
9    int P;
10   int Q;

12   ComplexPlot(DataIn din){
13     this.din=din;
14     P=din.getXDim();
15     Q=din.getYDim();
16   }

18   double sgn(double x){
19     return x>0 ? 1 : (x<0 ? -1 : 0);
20   }

22   BufferedImage refPlot(){
23     double xm=din.getXm();
24     double ym=din.getYm();
25     double xM=din.getXM();
26     double yM=din.getYM();
27     double mx=(xM-xm)/P;
28     double my=(yM-ym)/Q;
29     BufferedImage bi=new BufferedImage(P,Q,BufferedImage.TYPE_3BYTE_BGR);
30     Graphics gh=bi.getGraphics();
31     Complex w;
32     double u,v,s,R,R2,xx,yy;
33     float b,g,r;
34     for(int p=0;p<P;p++){
35       for(int q=0;q<Q;q++){
36         u=xm+mx*p;
37         v=yM-my*q;
38         R=Math.sqrt(u*u+v*v);
39         R2=R*R+1;
40         s=sgn(R-1)*(0.5-R/R2);
41         xx=u/R2/Math.sqrt(6);
42         yy=v/R2/Math.sqrt(2);
43         r=(float)(0.5+s+2*xx);
44         g=(float)(0.5+s-xx+yy);
45         b=(float)(0.5+s-xx-yy);
46         gh.setColor(new Color(r,g,b));
47         gh.fillRect(p,q,1,1);
48       }
49     }
50     gh.setColor(Color.BLACK);
51     gh.drawLine(0,Q/2,P,Q/2);
```

```
52      gh.drawLine(P/2,0,P/2,Q);
53      gh.drawLine(0,0,P,Q);
54      gh.drawLine(0,Q,P,0);
55      gh.setColor(Color.GREEN);
56      gh.drawOval(P/4,Q/4,P/2,Q/2);
57      gh.drawOval(3*P/8,3*Q/8,P/4,Q/4);
58      gh.drawOval(P/8,Q/8,3*P/4,3*Q/4);
59      return bi;
60    }

62    BufferedImage fctPlot(){
63      double xm=din.getXm();
64      double ym=din.getYm();
65      double xM=din.getXM();
66      double yM=din.getYM();
67      double mx=(xM-xm)/P;
68      double my=(yM-ym)/Q;
69      BufferedImage bi=new BufferedImage(P,Q,BufferedImage.TYPE_3BYTE_BGR);
70      Graphics gh=bi.getGraphics();
71      Complex w;
72      double x=0,y=0,u,v,s,R,R2,xx,yy;
73      float b,g,r;
74      for(int p=0;p<P;p++){
75        for(int q=0;q<Q;q++){
76          x=xm+mx*p;
77          y=yM-my*q;
78          w=din.fct(x,y);
79          u=w.re();
80          v=w.im();
81          R=Math.sqrt(u*u+v*v);
82          R2=R*R+1;
83          s=sgn(R-1)*(0.5-R/R2);
84          xx=u/R2/Math.sqrt(6);
85          yy=v/R2/Math.sqrt(2);
86          r=(float)(0.5+s+2*xx);
87          g=(float)(0.5+s-xx+yy);
88          b=(float)(0.5+s-xx-yy);
89          gh.setColor(new Color(r,g,b));
90          gh.fillRect(p,q,1,1);
91        }
92      }
93      int vx,vy;
94      int[] d={P,P,(int)(P*Math.sqrt(2)),(int)(P*Math.sqrt(2)),1000,1000,1000};
95      for(int k=0;k<d.length;k++){
96        for(int p=0;p<d[k];p++){
97          switch(k){
98            case 0: x=xm+mx*p;
99                    y=yM-my*Q/2;
100                   break;
101           case 1: x=xm+mx*P/2;
102                   y=yM-my*p;
103                   break;
104           case 2: x=xm+mx*p;
105                   y=yM-my*(P-p);
106                   break;
107           case 3: x=xm+mx*p;
108                   y=yM-my*p;
109                   break;
110           case 4: u=P/2+P/4*Math.cos(2*Math.PI*p/d[4]);
111                   v=Q/2+P/4*Math.sin(2*Math.PI*p/d[4]);
112                   x=xm+mx*u;
113                   y=yM-my*v;
114                   break;
115           case 5: u=P/2+P/8*Math.cos(2*Math.PI*p/d[4]);
116                   v=Q/2+P/8*Math.sin(2*Math.PI*p/d[4]);
117                   x=xm+mx*u;
118                   y=yM-my*v;
```

```
119                     break;
120             case 6: u=P/2+3*P/8*Math.cos(2*Math.PI*p/d[4]);
121                     v=Q/2+3*P/8*Math.sin(2*Math.PI*p/d[4]);
122                     x=xm+mx*u;
123                     y=yM-my*v;
124                     break;
125           }
126           if(k<4)
127             gh.setColor(Color.BLACK);
128           else
129             gh.setColor(Color.GREEN);
130           w=din.fct(x,y);
131           if(!w.isNaN()){
132             u=w.re();
133             v=w.im();
134             vx=(int)((u-xm)/mx);
135             vy=(int)((yM-v)/my);
136             if((0<=vx)&&(vx<=P)&&(0<=vy)&&(vy<=Q)){
137               gh.fillRect(vx,vy,1,1);
138             }
139           }
140         }
141       }
142       return bi;
143     }
144 }
```

Datele problemei, adică funcţia complexă f, domeniul D şi parametrii ferestrei P, Q sunt reţinute în clasa *DataIn*

```
1  package complexplot;
2  import org.nfunk.jep.type.Complex;
3  import org.nfunk.jep.*;

5  public class DataIn {
6    private double xm,ym,xM,yM;
7    private String var;
8    private String expr;
9    private int P=500;
10   private int Q=500;

12   JEP parser=null;

14   public void setXm(double xm){
15     this.xm=xm;
16   }
17   public void setYm(double ym){
18     this.ym=ym;
19   }
20   public void setXM(double xM){
21     this.xM=xM;
22   }
23   public void setYM(double yM){
24     this.yM=yM;
25   }

27   public double getXm(){
28     return xm;
29   }
30   public double getYm(){
31     return ym;
32   }
33   public double getXM(){
34     return xM;
35   }
36   public double getYM(){
37     return yM;
```

```
38    }

40    public String getVar(){
41      return var;
42    }
43    public String getExpr(){
44      return expr;
45    }

47    public void setP(int P){
48      this.P=P;
49    }
50    public int getP(){
51      return P;
52    }
53    public void setQ(int Q){
54      this.Q=Q;
55    }
56    public int getQ(){
57      return Q;
58    }

60    DataIn(String var,String expr){
61      this.var=var;
62      this.expr=expr;
63      parser=new JEP();
64      parser.addStandardFunctions();
65      parser.addStandardConstants();
66      parser.addComplex();
67      parser.addVariable(var,0,0);
68      parser.parseExpression(expr);
69    }

71    public Complex fct(double x,double y){
72       parser.addVariable(var,x,y);
73       return parser.getComplexValue();
74    }
75  }
```

Panourile sunt afişate de câte o instanţă a clasei *PlotPanel*

```
1  package complexplot;
2  import java.awt.*;
3  import java.text.DecimalFormat;

5  public class PlotPanel extends javax.swing.JPanel {

7    public PlotPanel(DataIn din, Image image, String name) {
8      int P=din.getP();
9      int Q=din.getQ();
10     initComponents();
11     jLabelName.setText(name);
12     DecimalFormat f=new DecimalFormat("#0.0");
13     String txt=f.format(din.getXm());
14     jLabelXm.setText(txt);
15     txt=f.format(din.getXM());
16     jLabelXM.setText(txt);
17     txt=f.format(din.getYm());
18     jLabelYm.setText(txt);
19     txt=f.format(din.getYM());
20     jLabelYM.setText(txt);
21     MyCanvas mc=new MyCanvas(image);
22     jPanelCanvas.setPreferredSize(new Dimension(din.getP(),din.getQ()));
23     jPanelCanvas.setLayout(new BorderLayout());
24     jPanelCanvas.add(mc,BorderLayout.CENTER);
25   }
```

```
27    private void initComponents() {
28      // cod generat de Netbeans
29    }

31    private javax.swing.JLabel jLabelName;
32    private javax.swing.JLabel jLabelXM;
33    private javax.swing.JLabel jLabelXm;
34    private javax.swing.JLabel jLabelYM;
35    private javax.swing.JLabel jLabelYm;
36    private javax.swing.JPanel jPanelCanvas;
37  }
```

Clasa *MyCanvas* este cea utilizată în secţiunea 7.1.
Aplicaţia este gestionată de clasa *Main*

```
1  package complexplot;
2  import java.awt.image.BufferedImage;
3  import java.awt.*;
4  public class Main extends javax.swing.JFrame {

6    public Main() {
7        initComponents();
8    }


11   private void initComponents() {
12     // cod generat de Netbeans
13   }

15   private void jButtonComputeMouseClicked(java.awt.event.MouseEvent evt)
16     String var=jTextFieldVar.getText();
17     String expr=jTextFieldExpr.getText();
18     DataIn din=new DataIn(var,expr);
19     int P=400;
20     int Q=400;
21     double val=(new Double(jTextFieldXm.getText())).doubleValue();
22     din.setXm(val);
23     val=(new Double(jTextFieldYm.getText())).doubleValue();
24     din.setYm(val);
25     val=(new Double(jTextFieldXM.getText())).doubleValue();
26     din.setXM(val);
27     val=(new Double(jTextFieldYM.getText())).doubleValue();
28     din.setYM(val);
29     din.setP(P);
30     din.setQ(Q);
31     ComplexPlot cp=new ComplexPlot(din);
32     BufferedImage bf0=cp.refPlot();
33     PlotPanel p0=new PlotPanel(din,(Image)bf0,din.getVar()+" -> " +
34       din.getVar());
35     BufferedImage bf=cp.fctPlot();
36     PlotPanel pf=new PlotPanel(din,(Image)bf,din.getVar()+" -> " +
37       din.getExpr());
38     String title="Reprezentarea functiei compleze ";
39     javax.swing.JFrame jframe = new javax.swing.JFrame(title);
40     jframe.addNotify();
41     jframe.getContentPane().setLayout(new GridLayout(1,2));
42     jframe.getContentPane().add(p0);
43     jframe.getContentPane().add(pf);
44     jframe.setSize(2*P+120,Q+100);
45     jframe.setVisible(true);
46   }

48   public static void main(String args[]) {
49     java.awt.EventQueue.invokeLater(new Runnable() {
50       public void run() {
51         new Main().setVisible(true);
52       }
```

```
53      });
54    }

56    private javax.swing.JButton jButtonCompute;
57    private javax.swing.JLabel jLabelExpr;
58    private javax.swing.JLabel jLabelVar;
59    private javax.swing.JLabel jLabelXM;
60    private javax.swing.JLabel jLabelXm;
61    private javax.swing.JLabel jLabelYM;
62    private javax.swing.JLabel jLabelYm;
63    private javax.swing.JTextField jTextFieldExpr;
64    private javax.swing.JTextField jTextFieldVar;
65    private javax.swing.JTextField jTextFieldXM;
66    private javax.swing.JTextField jTextFieldXm;
67    private javax.swing.JTextField jTextFieldYM;
68    private javax.swing.JTextField jTextFieldYm;
69  }
```

Cu această aplicaţie se obţin următoarele rezultate:

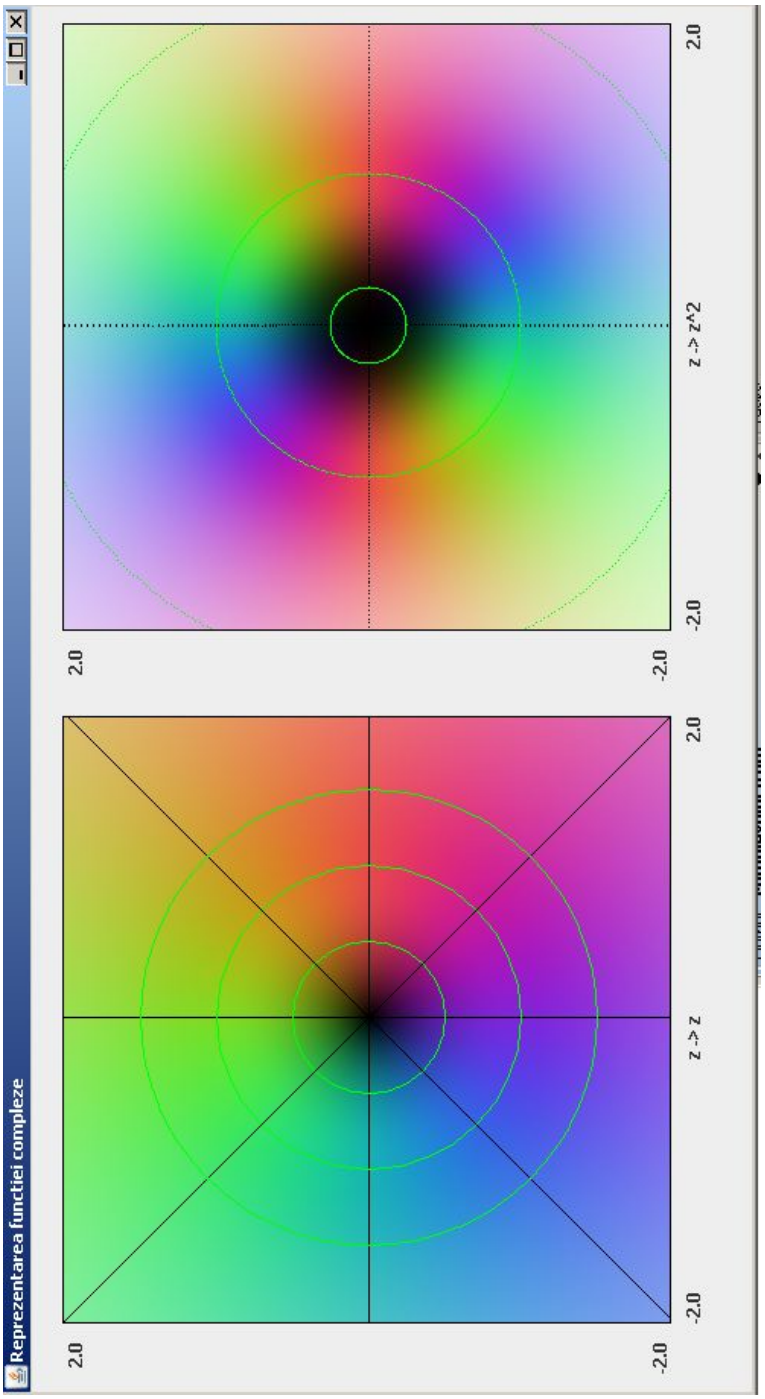


Fig. 7.10: Vizualizarea funcţiei $f(z) = z^2$.

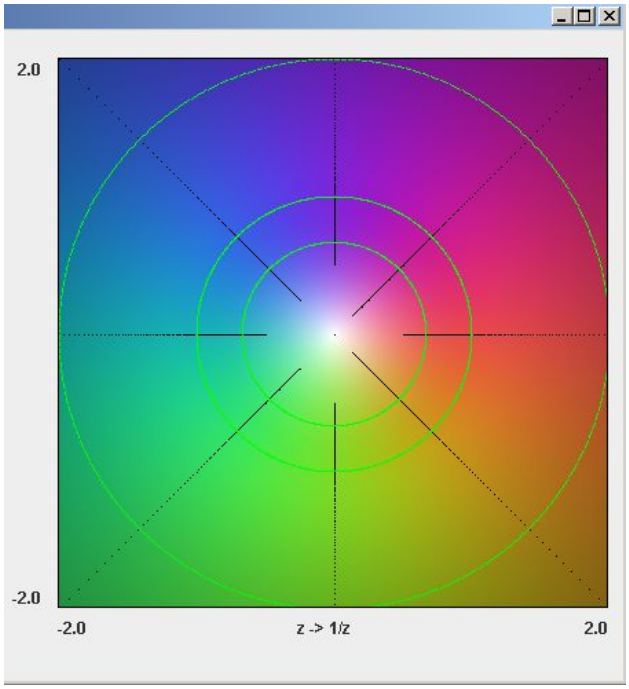


Fig. 7.11: Vizualizarea funcţiei $f(z) = \frac{1}{z}$.

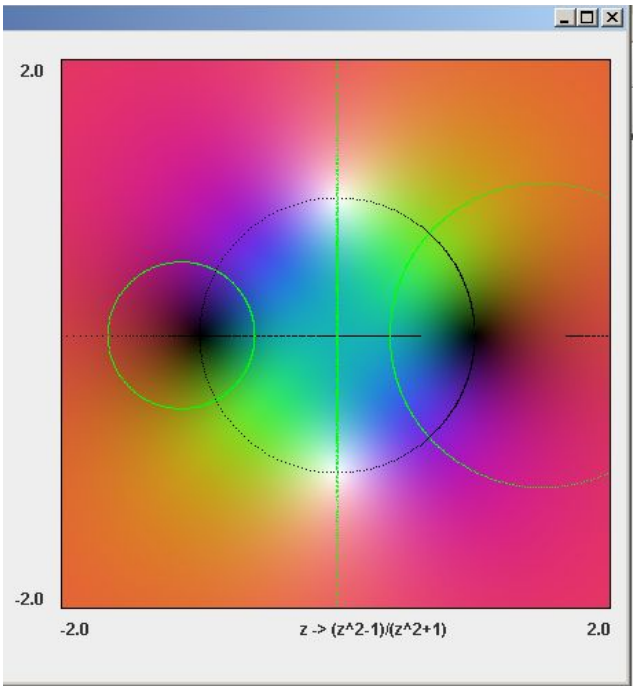


Fig. 7.12: Vizualizarea funcţiei $f(z) = \frac{z^2-1}{z^2+1}$.

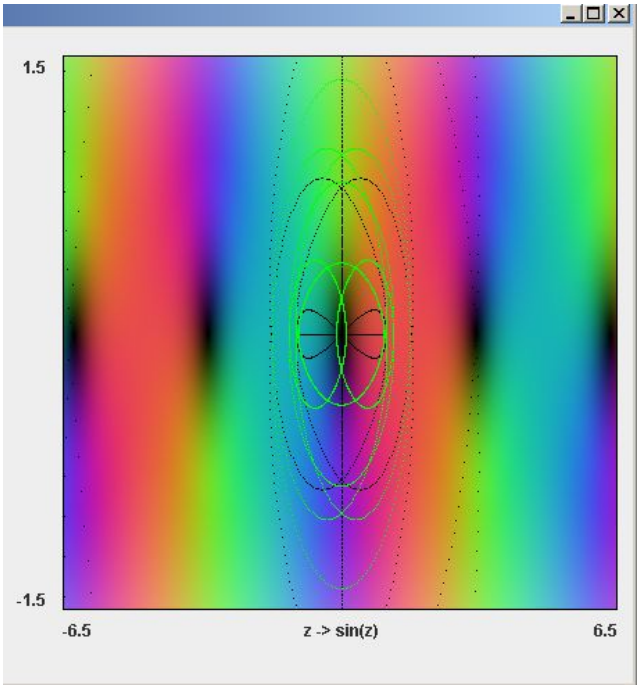


Fig. 7.13: Vizualizarea funcţiei $f(z) = \sin z$.

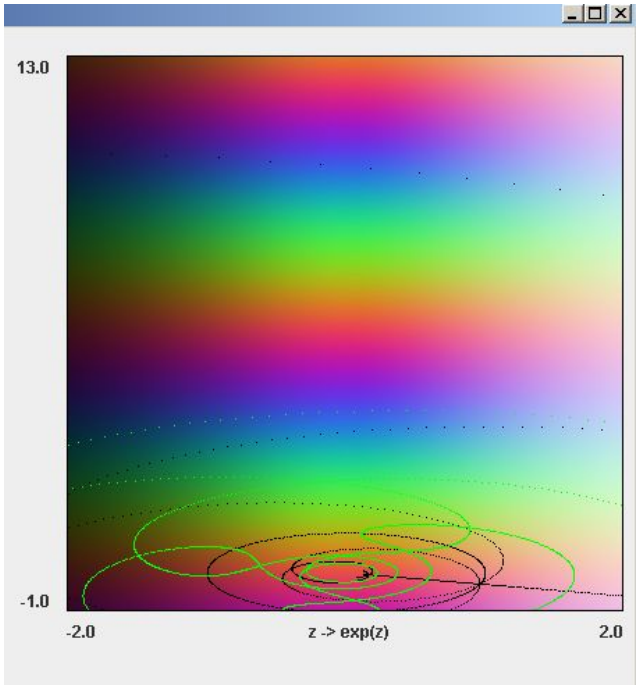


Fig. 7.14: Vizualizarea funcţiei $f(z) = \exp z$.

Capitolul 8

Aplicaţii Web

Reţelele locale, internetul, răspândirea pe o arie geografică a resurselor şi a locaţiilor în care se petrec acţiuni ce ţin de o activitate bine definită sau sunt urmărite, gestionate din alte locuri au drept consecinţă existenţa aplicaţiilor distribuite. Termenul distribuit se referă tocmai la faptul că componente ale aplicaţiei se află pe calculatoare diferite, dar între care au loc schimburi de date. Dacă părţile unei aplicaţii sau resursele utilizate se găsesc pe calculatoare distincte atunci aplicaţia se numeşte distribuită.

Între părţile sau resursele unei aplicaţii distribuite au loc schimburi de date la realizarea cărora concură sistemul de calcul, sistemul de operare şi limbajul de programare.

Astfel, se vorbeşte de programare distribuită ca mijloc de realizare a aplicaţiilor distribuite.

Schimburile de date se pot realiza prin mai multe metode. Punem în evidenţă două modele de aplicaţii distribuite:

- **client-server**: Programul server execută cererile clienţilor.

  Printre aplicaţiile distribuite de tip client-server, în care comunicaţiile se bazează pe protocolul *http,* se disting

  - *Aplicaţii Web (site):* cererea adresată serverului este lansată de o persoană prin intermediul unui site, utilizând un program navigator (browser Web).
  - *Servicii Web:* cererea către server se face de un program.

- **dispecer-lucrător**: Programul dispecer distribuie sarcinile de executat lucrătorilor şi le coordonează activitatea.

Limitându-ne la aplicaţii distribuite de tip client-server şi la platforma Java, în acest capitol se vor utiliza următoarele tehnologii de programare:

- *servlet*-ul, care reprezintă tehnologia de baza pentru realizarea aplicaţiilor Web în Java;
- *websocket* bazat pe protocolul omonim, introdus de HTML5.
- *Google Web Toolkit* (GWT) cadru de lucru pentru dezvoltarea aplicaţiilor Web.

Scopul urmărit este adaptarea programelor de calcul ştiinţific la aplicaţii Web.
Componenta server a unei aplicaţii Web

- conţine o clasă Java care se excută de un server Web, compatibil;
- este gestionată de serverul Web;
- este capabilă să recepţioneze şi să răpundă cererilor formulate de clienţi.

Structura minimală a unei aplicaţii Web este

```
catalogAplicatiei
|--> WEB-INF
|    |--> classes
|    |    |    *.class
|    |--> lib
|    |    |    *.jar
|    index.html
```

Catalogul `classes` conţine fişierele *class* ale aplicaţiei Web.
Catalogul `lib` este opţional şi va conţine resursele *jar* suplimentare cerute de clasele aplicaţiei Web.
Prin intermediul fişierului *index.html* se apelează aplicaţia Web. Acest fişier este totodată şi client Web. Adresa de apelare (URL - *Universal Resource Locator*) a aplicaţiei Web este

`http://`*host:port/catalogAplicaţieiWeb*

Dacă în loc de *index* fişierul `html` de apelare are alt nume, de exemplu *xyz.html* atunci adresa de apelare va fi

`http://`*host:port/catalogAplicaţieiWeb/xyz.html*

host este numele calculatorului pe care rulează serverul Web - *gazda* aplicaţiei Web. Portul implicit utilizat de un server Web container de servlet este 8080.
Catalogul aplicaţiei este denumit *context*-ul aplicaţiei Web.
Programarea şi utilizarea unei aplicaţii Web necesită:

- Cunoaşterea elementului (marcaj, *tag*) `html` <`form`> pentru realizarea formularelor de introducere a datelor;
- Utilizarea unui server Web, container de servleţi. Dintre produsele gratuite amintim: *apache-tomcat, jetty, glassfish.*

8.1 Servlet

Interfaţa de programare (API) pentru servlet nu face parte din `JDK`, fiind implementat de fiecare producător de server Web container de servlet.

Legătura dintre serverul Web cu clasa servlet-ului se poate realiza

- programat – prin adnotări[1] în codul servlet-ului;
- descriptiv – în catalogul `WEB-INF` se editează fişierul `web.xml`.

  În versiunile anterioare versiunii 3.0 ale interfeţei de programare servlet aceasta a fost unica opţiune.

Trebuie demarcată diferenţa dintre apelarea / lansarea în execuţie a clasei servlet de apelarea aplicaţiei Web.

Modul programat se bazează pe adnotarea `javax.servlet.annotation.WebServlet` cu elementele:

`String`	name
`String[ ]`	urlPatterns
`@InitParams[ ]`	initParam
`boolean`	asyncSupported
`long`	asyncTimeout

Modul descriptiv În fişierul `web.xml` apar elementele

1. <`servlet`> leagă *numele servlet-ului* definit în elementul <`servlet-name`> de clasa servlet-ului dat în elementul <`servlet-class`> .
2. <`servlet-mapping`> defineşte numele sub care servlet-ul identificat prin <`servlet-name`> *nume servlet*< /`servlet`> se invocă din programul navigator. Acest identificator - *numeApel* - se fixează în elementul <`url-pattern`> . Identificatorul are ca prefix caracterul / (slash).

Structura unui fişier `web.xml` este

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <servlet>
      <servlet-name>nume_servlet_1</servlet-name>
      <servlet-class>Nume_clasa</servlet-class>
   </servlet>
   <servlet>
      <servlet-name>nume_servlet_2</servlet-name>
      <servlet-class>Nume_clasa</servlet-class>
   </servlet>
   . . .
   <servlet-mapping>
      <servlet-name>nume_servlet_1</servlet-name>
```

[1] În Java o adnotare (*annotation*) este o metadata a unui element de cod (identificator al unei entităţi din codul Java).

O adnotare poate să-şi facă efectul asupra codului sursš, înaintea compilării, asupra codului obiect, după compilare, dar înaintea executării sau în timpul execuţiei codului.

```
15      <url-pattern>/numeApel_1</url-pattern>
16    </servlet-mapping>
17    <servlet-mapping>
18      <servlet-name>nume_servlet_2</servlet-name>
19      <url-pattern>/numeApel_2</url-pattern>
20    </servlet-mapping>
21    . . .
22 </web-app>
```

Unui element <`servlet`> îi pot corespunde mai multe elemente <`servlet-mapping`>, prin utilizarea de *numeApel* diferite.

Opţional `web.xml` poate conţine elementul

```
<welcome-file-list>
  <welcome-file>
    fisier.html sau jsp
  </welcome-file>
</welcome-file-list>
```

cu precizarea fişierelor html sau jsp care apelează aplicaţia Web. Declaraţia fişierului *index.html* este implicită.

Compilarea clasei servlet necesită completarea variabilei de mediu `classpath` cu fişierul `TOMCAT_HOME\lib\servlet-api.jar`.

Odată completată structura de cataloage şi fişiere ale aplicaţiei servlet această structură trebuie copiată în catalogul `TOMCAT_HOME\webapps`. Această operaţie se numeşte **desfăşurarea** (*deployment*) sau instalarea servlet-ului. Copierea se poate executa şi cu serverul Web pornit.

Pentru instalarea unui servlet există mai multe alternative:

- Din catalogul *catalogAppServlet* se realizează arhiva *catalogAppServlet.war*

  `jar cfv` *catAppServlet.war* `WEB-INF\* index.html`

  care se copiază în catalogul *TOMCAT_HOME\webapps.*

  Serverul Web *tomcat* va dezarhiveaza arhiva. Astfel servlet-ul este instalat.

  Această instalare se numeşte *instalare dinamică - hot deployment.*

  Dacă fişierul `war` este creat, atunci în locul copierii, instalarea se poate face prin componenta *manager* a lui *tomcat.*

- O altă posibilitate de instalare a unui servlet în serverul web *tomcat* este prin intermediul produsului *apache-tomcat-deployer. apache-tomcat-deployer* permite instalarea unui servlet de la distanţă cât şi instalarea comandată dintr-un program.

8.1.1 Codul unui servlet

Un servlet implementează interfaţa `Servlet` sau extinde una din clasele `GenericServlet` sau `HttpServlet`. `GenericServlet` implementează interfaţa `Servlet`, iar `HttpServlet` extinde clasa `GenericServlet`. Extinzând clasa `GenericServlet` nu este nevoie de rescrierea tuturor metodelor abstracte ale intrefeţei `Servlet`.

Uzual clasa unui servlet va fi o clasă care extinde clasa `HttpServlet`, programatorul va suprascrie metodele `doGet(...)` sau `doPost(...)`, în funcţie de metoda utilizată de client la lansarea cererii.

Practic, un servlet constă din scrierea metodelor

- `void init(ServletConfig` *config*`)`

  Această metodă este opţională.

  ```
  public void init(ServletConfig config) throws ServletException{
    super.init(config);
    // cod de initializare
  }
  ```

  Obiectul *config* are o metodă `String getInitParameter(String` *numeParam*`)` cu ajutorul căreia se pot recupera parametri de initializare asociaţi servlet-ului şi care se dau fie prin adnotarea `@initParams`, fie în fişierul `web.xml` prin elementele

  ```
  <init-param>
    <param-name> NumeleParametrului </param-name>
    <param-value> Valoare </param-value>
  </init-param>
  ```

  cuprinse în elementul <`servlet`>.

- `protected void doGet(HttpServletRequest` *req,* `HttpServletResponse` *res*`) throws IOException, ServletException`

  Tratează o cerere trimisă cu metoda GET (vezi marcajul <form>).

- `protected void doPost(HttpServletRequest` *req,* `HttpServletResponse` *res*`) throws IOException, ServletException`

  Tratează o cerere trimisă cu metoda POST (vezi marcajul <form>).

Activităţile de întreprins într-o metodă `doGet()` sau `doPost()` sunt

1. Stabilirea naturii răspunsului:
   `res.setContentType(String` *tip*`)`

   unde *tip* specifică tipul *MIME - Multipurpose Internet Mail Extensions* al răspunsului:

   - `"text/html"` - pagină html;
   - `"text/xml"` - document xml;
   - `"text/plain"` - text;
   - `"image/jpg"` - imagine gif;
   - `"image/gif"` - imagine jpg.

2. Se obţine o referinţa către un obiect care realizeaza transmisia datelor către navigatorul clientului:
   `ServletOutputStream out = res.getOutputStream();`
   sau
   `PrintWriter out=res.getWriter();`

3. Se preiau datele cererii cu una din metodele:
   `String getParameter(String` *numeParapetru*`)`
   `java.util.Enumeration getParameterNames()`

4. Rezolvă cererea clientului;

5. Formează şi *scrie* răspunsul;

6. Închide conexiunea obiectului prin care s-a realizat transmisia datelor către navigatorul clientului prin `out.close()`.

Un utilizator lansează o cerere către servlet. De obicei acest lucru se realizează prin clientul Web. Programul navigator trimite cererea serverului Web prin intermediul căruia este lansat clasa servlet-ului în acţiune.

Versiunea 3.0 a interfeţei de programare pentru servlet oferă posibilitatea programării asincrone a actiunii servletului prin includerea acestuia într-un fir de execuţie.

Mai mult în versiunea 3.1, acţiunea firului de execuţie este înlocuită prin programarea unor interfeţe de tip *listener.*

Ciclul de viaţă al unui servlet. Când un servlet este apelat prima dată de către serverul Web se execută metoda `init`. După aceasta, fiecărei cereri lansate de un utilizator i se asociază un fir de execuţie în care se apelează metoda `service`. Metoda `service` apelează apoi metodele `doGet(), doPost()`.

Exemplul 8.1.1 *Servlet pentru calculul unei integrale.*

Transformăm aplicaţia dezvoltată în (1.2) într-un servlet.
Codul servlet-ului va fi

```
1  package integrala;
2  import java.io.IOException;
3  import java.io.PrintWriter;
4  import javax.servlet.ServletException;
5  import javax.servlet.http.HttpServlet;
6  import javax.servlet.http.HttpServletRequest;
7  import javax.servlet.http.HttpServletResponse;
8  import javax.servlet.annotation.WebServlet;
9  import mathlib.client.cvadra.*;
10 import mathlib.client.cvadra.impl.*;

12 @WebServlet(urlPatterns = "/integ")

14 public class MetodaSimpsonServlet extends HttpServlet{

16   public void doGet(HttpServletRequest req,HttpServletResponse res)
17       throws ServletException,IOException{
18     String a=req.getParameter("a");
19     String b=req.getParameter("b");
```

```
20      String var=req.getParameter("svar");
21      String expr=req.getParameter("expr");
22      String eps=req.getParameter("eps");
23      String nmi=req.getParameter("nmi");
24      String tip=req.getParameter("tip");
25      System.out.println(var+" "+expr+" "+a+" "+b+" "+eps+" "+nmi+" "+tip);
26      JepDataIn din=new JepDataIn(var,expr);
27      din.setA(a);
28      din.setB(b);
29      din.setEps(Double.parseDouble(eps));
30      din.setNmi(Integer.parseInt(nmi));
31      IMetodaSimpson obj=new MetodaSimpsonWeb();
32      DataOut dout=obj.metodaSimpson(din);
33      res.setContentType(tip);
34      PrintWriter out=res.getWriter();
35      if(tip.equals("text/html")){
36        out.println("<html>");
37        out.println("<head><title> </title></head>");
38        out.println("<body bgcolor=\"#bbccbb\">");
39        out.println("<center>");
40        out.println("<h1>Calculul integralei prin metoda Simpson </h1>");
41        out.println("<p>");
42        out.println( "Indicatorul de raspuns : "+ dout.getInd());
43        out.println("</p>");
44        out.println("<p>");
45        out.format( "Integrala : %1$12.6f",dout.getIntegrala());
46        out.println("</p>");
47        out.println("<p>");
48        out.println( "Numarul iteratiilor efectuate : "+ dout.getNi());
49        out.println("</p>");
50        out.println("<br/>");
51        out.println("</center>");
52        out.println("</body>");
53        out.println("</html>");
54      }
55      else{
56        out.println("Indicatorul de raspuns : "+dout.getInd());
57        out.println("Integrala : "+dout.getIntegrala());
58        out.println("Numarul iteratiilor efectuate : "+dout.getNi());
59      }
60      out.close();
61    }

63    public void doPost(HttpServletRequest req,HttpServletResponse res)
64       throws ServletException,IOException{
65     doGet(req,res);
66    }
67  }
```

Pentru fixarea naturii răspunsului `text/html` sau `text/plain` s-a introdus variabila *tip,* care în fişierul de invocare *index.html* primeşte *pe ascuns* valoarea `text/html`. În cazul în care vom apela servlet-ul dintr-un program, va fi mai avantajos să primim răspunsul ca `text/plain`.

Desfăşurarea servlet-ului este

```
appinteg
|--> WEB-INF
|    |--> classes
|    |    |--> integrala
|    |    |    |    MetodaSimpsonServlet.class
|    |    |--> lib
|    |    |    |    mathlib.jar
|    |    |    |    jep-2.4.1.jar
|    index.html
```

Codul clientului Web, adică fişierului *index.html,* este:

```
<!doctype html>
  <body bgcolor="#bbccbb">
    <center>
    <h1> Pagina de apelare a servlet-ului MetodaSimpson </h1>
    <p> <h3> Calculul unei integrale </h3>
    <p>Introduceti:

    <form method="post"
          action="/appinteg/integ">
      <table border="2" >
        <tr>
          <td> Variabila de integrare : </td>
          <td> <input type="text" name="svar" size=30 required> </td>
        </tr>
        <tr>
          <td> Expresia functiei de integrat : </td>
          <td> <input type="text" name="expr" size=30 required> </td>
        </tr>
        <tr>
          <td> Limita inferioara a intervalului : </td>
          <td> <input type="text" name="a" size=30 required> </td>
        </tr>
        <tr>
          <td> Limita superioara a intervalului : </td>
          <td> <input type="text" name="b" size=30 required> </td>
        </tr>
        <tr>
          <td> Toleranta : </td>
          <td> <input type="text" name="eps" value="1.0e-8" size=30 required> </td>
        </tr>
        <tr>
          <td> Numar maxim admis de iteratii : </td>
          <td> <input type="text" name="nmi" value="50" size=30 required> </td>
        </tr>
        <tr>
          <td align="center"> <input type="submit" value="Calculeaza"> </td>
          <td>  </td>
        </tr>
      </table>
      <input type="hidden" name="tip" value="text/html" >
    </form>
    </center>
  </body>
</html>
```

Pe lângă un client Web se pot realiza şi programe client.

8.1.2 Program client al unui servlet

Apelarea unui servlet dintr-un program Java – adică lansarea unei cereri şi recepţionarea răspunsului furnizat de servlet – se poate obţine cu produsul *httpcomponents-client* dezvoltat de *apache.*

Într-un asemenea caz, din punctul de vedere al clientului este mai avantajos ca răspunsul servlet-ului fie `text/plain,` în loc de `text/html.`

Există mai multe modalităţi de programare dintre care vom utiliza varianta de programare fluentă.

Exemplul 8.1.2 *Program client pentru servlet-ul MetodaSimpson (8.1.1):*

```
1  package integrala;
2  import java.util.Scanner;
3  import org.apache.http.client.fluent.Form;
4  import org.apache.http.client.fluent.Request;

6  public class Client{
7    static String url = "http://localhost:8080/appinteg/integ";

9    public static void main(String[] args) {
10     Scanner scanner=new Scanner(System.in);
11     System.out.println("Introduceti:");
12     System.out.println("Variabila ");
13     String svar=scanner.next();
14     System.out.println("Expresia de integrat ");
15     String expr=scanner.next();
16     System.out.println("Limita inferioara ");
17     double a=scanner.nextDouble();
18     System.out.println("Limita superioara ");
19     double b=scanner.nextDouble();
20     System.out.println("Toleranta ");
21     double eps=scanner.nextDouble();
22     System.out.println("Numar maxim admis de iteratii ");
23     int nmi=scanner.nextInt();
24     try{
25       String result=Request.Post(url)
26         .bodyForm(Form.form()
27           .add("svar",svar)
28           .add("expr",expr)
29           .add("a",new Double(a).toString())
30           .add("b",new Double(b).toString())
31           .add("eps",new Double(eps).toString())
32           .add("nmi",new Integer(nmi).toString())
33           .add("tip","text/plain")
34           .build())
35         .execute().returnContent().asString();
36       System.out.println(result);
37     }
38     catch(Exception e){
39       e.printStackTrace();
40     }
41   }
42 }
```

8.1.3 Dezvoltarea unui servlet prin *maven*

Dorim să prezentăm dezvoltarea aplicaţiei servlet prin intermediul lui *maven.* Dezvoltarea servlet-ului revine la parcurgerea paşilor:

1. Generarea aplicaţiei:

```
set GroupID=integrala
set ArtifactID=appinteg
set Version=1.0
mvn -B archetype:generate
  -DgroupId=%GroupID%
  -DartifactId=%ArtifactID%
  -Dversion=%Version%
  -DarchetypeArtifactId=maven-archetype-webapp
```

2. Se adaptează structura de cataloage şi fişiere la

```
appinteg
|-- src
|   |-- main
|   |   |-- java
|   |   |   |-- integrala
|   |   |   |   |   MetodaSimpsonServlet.java
|   |   |-- resources
|   |   |-- webapp
|   |   |   |-- WEB-INF
|   |   |   |   |-- lib
|   |   |   |   |   |   mathlib.jar
|   |   |   |   |   |   jep-2.4.1.jar
|   |   |   |   |   web.xml
|   |   |   |   index.html
|   pom.xml
```

3. Fişierul `pom.xml` se completează cu

   - Referinţa la `javax.servlet.servlet-api` necesară compilării,

```
 <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>3.0.1</version>
</dependency>
```

   - Referinţele către pachetele *mathlib.client.cvadra, mathlib.client.cvadra.impl, jep* aflate în depozitul local *maven*

```
<dependency>
    <groupId>mathlib.client.cvadra</groupId>
    <artifactId>icvadra</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>mathlib.client.cvadra.impl</groupId>
    <artifactId>cvadraimpl</artifactId>
    <version>1.0</version>
</dependency>
<dependency>
    <groupId>jep</groupId>
    <artifactId>jep</artifactId>
    <version>2.4.1</version>
</dependency>
```

4. Deoarece s-a utilizat modul programat la programarea servletului - introdus de *servlet-api* 3.0 - fişierul `web.xml` trebuie înlocuit cu

```
1 ?xml version="1.0" encoding="UTF-8"?>
2 <web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4 </web-app>
```

5. Prelucrarea revine la `mvn clean package`

   Arhiva `war` creată în catalogul `target` al proiectului se poate desfăşura în orice server Web.

Avem posibilitatea să verificăm aplicaţia din *maven:*

1. Se completează fişierul *pom.xml* în elementul <`build`> cu

```
<plugins>
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>9.3.6.v20151106</version>
      </plugin>
</plugins>
```

   Versiunea trecută se poate actualiza.

2. Se modifică în fişierul *index.html* al aplicaţiei servlet atributul `action` în *integ* şi se reia comanda `mvn clean package`.

3. Se lansează serverul Web *jetty* `mvn jetty:run`

4. Dintr-un navigator, aplicaţia se apelează `http://localhost:8080`.

8.1.4 Servlet ca modul OSGi

Integrarea unei aplicaţii Web într-o platformă *OSGi* necesită o abordare specifică. Integrată într-o platformă *OSGi,* aplicaţia Web nu mai este desfăşurată nemijlocit în serverul Web, dar apelurile se vor adresa în continuare serverului Web.

Integrarea unei aplicaţii servlet într-o componentă *OSGi* se va baza pe interfaţa *org.osgi.service.http.HttpService,* pentru care vom folosi implemantarea `org.apache.felix.http.bundle-*.jar`. Cu foarte puţine diferenţe, în varianta de programare pe care o prezentă, componenta *OSGi* se va putea utiliza pe platformele *OSGi apache-karaf* şi *glassfish.*

Interfaţa `HttpService` declară metodele

- `void registerResources(String` *alias,* `String` *name,* `HttpContext` *context*`) throws NamespaceException`
- `void registerServlet(String` *alias,* `Servlet` *servlet,* `Dictionary` *initparams,* `HttpContext` *context*`)ServletException, NamespaceException`
- `void unregister(String` *alias*`)`

Structura componentei OSGi corespunzătoare unui servlet este

```
|--> META-INF
|    |    MANIFEST.MF
|    ClasaServlet.class
|    Activator.class
|    fisier.html
```

Rămâne la latitudinea programatorului să includă sau nu pagina `html`.

Clasa servlet-ului rămâne nemodificată iar în clientul Web reprezentat de fişierul `html` doar valoarea atributului `action` diferă:

Cadrul *OSGi*	Apel (aplicaţie, servlet)
apache-karaf	*http://host:8080/fişier.html /numeApel*
glassfish	*http://host:8080/osgi/fişier.html /osgi/numeApel*

Exemplul 8.1.3 *Clasa activatorului pentru servletul MetodaSimpson 8.1.1.*

```
1  import org.osgi.framework.BundleActivator;
2  import org.osgi.framework.BundleContext;
3  import org.osgi.framework.ServiceReference;
4  import org.osgi.service.http.HttpService;

6  public class Activator implements BundleActivator{

8    public void start(BundleContext context) throws Exception{
9      ServiceReference sRef = context.getServiceReference(HttpService.class.getName());
10     if (sRef != null){
11       HttpService service = (HttpService) context.getService(sRef);
12       service.registerServlet("/integ",new integrala.MetodaSimpsonServlet(),null,null);
13       service.registerResources("/appinteg", "/index.html", null);
14     }
15   }

17    public void stop(BundleContext context) throws Exception{}
18 }
```

În acest caz structura modulului OSGi este

```
|-->integrala
|    |    MetodaSimpsonServlet.class
|-->lib
|    |    mathlib.jar
|    |    jep-2.4.1..jar
|--> META-INF
|    |    MANIFEST.MF
|    Activator.class
|    index.html
```

unde MANIFEST.MF este

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: MetodaSimpsonServlet
4  Bundle-SymbolicName: appinteg
5  Bundle-Version: 1.0.0
6  Bundle-Activator: Activator
7  Bundle-Localization: plugin
8  Import-Package: javax.servlet,javax.servlet.http,org.osgi.framework;version="1.3.0",
9    org.osgi.service.http;version="1.2.0"
10 Bundle-Classpath: .,lib/mathlib.jar,lib/jep-2.4.1.jar
```

Detalii de operare - *apache-karaf*

Lansarea produsului se poate prin fişierul de comenzi

```
set JAVA_HOME=. . .
set KARAF_HOME=. . .
del %KARAF_HOME%\lock
del %KARAF_HOME%\instances\*
rmdir %KARAF_HOME%\instances
%KARAF_HOME%\bin\karaf.bat clean
```

În catalogul %KARAF_HOME% se generează catalogul *instances* iar în fereastra DOS va apare prompt-ul `karaf@root>`.

Oprirea se obţine apăsând tastele CTRL+D.

Karaf posedă o consolă DOS dar şi o consolă Web.

Instalarea componentelor OSGi se poate face

- copiindu-le în catalogul `%KARAF_HOME%\deploy`
- în mod obişnuit, prin comanda `install file:...`

  În acest caz fişierul `MANIFEST.mf` trebuie să conţină atributele

  ```
  Bundle-ManifestVersion: 2
  Bundle-SymbolicName: numeComponentaOSGi
  ```

Comenzi OSGi uzuale sunt: `start` n, `stop` n, `install file:...`, `uninstall` n, `list`, `help`.

Mediul OSGi *apache-karaf* utilizează serverul Web incorporat *Jetty* pe portul 8181. Pentru a schimba portul se crează în prealabil fişierul `etc\org.ops4j.pax.web.cfg` cu conţinutul

`org.osgi.service.http.port`=8080

Trebuie instalat suportul pentru protocolul `http` prin

```
feature:install http
```

Apelarea unei aplicaţii servlet va fi *http://host:port/fişier.html*

Detalii de operare - *glassfish*

Mediul *OSGi* are la bază platforma *apache-felix.* După lansarea serverului de aplicaţii *glassfish* comenzile OSGi se apelează prin

`asadmin osgi` *comanda_OSGi*

Astfel pentru lansarea unui servlet se va executa comanda:

`asadmin osgi start file:.../`*arhiva*`.jar`

Apelarea aplicaţiei servlet va fi *http://host:port/osgi/fişier.html*

8.2 WebSocket

Protocolul websocket se iniţiază dintr-o comunicaţie bazată pe protocolul `http` prin mecanismul *upgrade,* introdus de *servlet-api* 3.0: Într-un mesaj `http` trimis de client se indică solicitarea de trecere de la protocolul `http` la protocolul WebSocketprin prezenţa antetului `Upgrade: websocket`. Exemplul unui asemenea mesaj este

```
GET /HelloWebSocket/hello HTTP/1.1
Upgrade: websocket
Connection: Upgrade
Host: localhost:9090
Origin: null
Sec-WebSocket-Key: tLZ8VGZ8Cw8kt0BvhuV6Vw==
Sec-WebSocket-Version: 13
Sec-WebSocket-Extensions: x-webkit-deflate-frame
Cookie: JSESSIONID=2BCFF666164139524DD92D573C3859F7;
 JSESSIONID=bf12d2417e8eb1bcd6da6137af9d;
 treeForm_tree-hi=treeForm:tree:applications
```

Mesajul de răspuns afirmativ este

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: ZDE5NmS4T9spIby8/vo/V+rbNAs=
```

Dacă între cei doi parteneri se stabileşte acordul (*handshake*) atunci restul comunicaţiilor au loc prin intermediul unui soclu TCP pe portul 80.

Astfel ciclul de viaţă al procesului de comunicaţie este:

1. Un client solicită printr-un mesaj `http` acordul pentru trecerea la protocolul WebSocket.
2. Serverul răspunde acceptând acordul.
3. Odată stabilită conexiunea, acesta devine bidirecţională (simetrică), clientul şi serverul transmit şi recepţionează mesaje.
4. Una din părţi închide conexiunea.

8.2.1 Interfaţa de programare HTML5 de client WebSocket

Interfaţa de programare este definită de un obiect JavaScript:
Constructor

- `WebSocket(in String` *uri*`)`

  unde *uri* este de forma `ws://`*host:port/context/numeApel*

Funcţii

- `attribute function onopen(`*evt*`)`
- `attribute function onmessage(`*evt*`)`
- `attribute function onerror(`*evt*`)`

- `attribute function onclose(`*evt*`)`
- `boolean send(in String` *data*`)`
- `close()`

Atribute

- `readyState`

  Starea conexiunii:

Valoare	Semnificaţia
0	Nu s-a stabilit conexiunea
1	Conexiune pregatită pentru comunicaţii
2	Conexiune în pragul *confirmării* (*handshake*)
3	Conexiune închisă şi nu mai poate fi redeschisă

- `bufferedAmount`

  Numărul octeţilor trimişi de funcţia `send`. Datele sunt codificate UTF-8.

Un şablon de utilizare poate fi

```
<script language="javascript" type="text/javascript">
     var wsUri = "ws://host:8080/context/numeApel";
     var websocket = new WebSocket(wsUri);
     websocket.onopen = function(evt) { . . . };
     websocket.onmessage = function(evt) { . . . };
     websocket.onerror = function(evt) { . . . };
     websocket.onclose = function(evt) { . . . };
  . . .
</script>
```

Expedierea datelor.
Considerăm formularul HTML

```
<form  name="myform">
  <input type="text" name="xyz" . . ./>
  . . .
  <input type="button" onclick="send"/>
</form>
```

Şablonul funcţiei *send* este

```
function send(){
   var xyz=document.myform.xyz.value;
   websocket.send(xyz);
}
```

Recepţia unui rezultat furnizat de server. Funcţia `onmessage` permite recuperarea rezultatului din `evt.data`.

8.2.2 WebSocket în Java

Interfaţa de programare Java pentru WebSocket declară clase atât pentru server cât şi pentru client.

Programarea serverului se poate realiza prin adnotări (*Annotation driven*) potrivit şablonului

```
@ServerEndpoint(value="/urlPattern")
public class EndpointWebSocketServer {
  private static Set<Session> sessions =
    Collections.synchronizedSet(new HashSet<Session>());

  @OnMessage
  public void myTask(String msg, Session session)
      throws IOException,EncodeException{
     . . .
  }

  @OnOpen
  public void onOpen(Session session){
    sessions.add(session);
  }

  @OnClose
  public void onClose(Session session){
    sessions.remove(session);
  }
}
```

Metoda cu adnotareae `@OnMessage` asigură accesul la datele cererii unui client.

Acest şablon necesită familiarizarea cu componente din interfaţa de programare WebSocket.

Interfaţa `javax.websocket.Session`

Metode

- `void addMessageHandler(MessageHandler` *handler*`)`
- `RemoteEndpoint.Basic getBasicRemote()`

  Prin intermediul unui obiect de tip `RemoteEndpoint.Basic` se programează expedierea răspunsului către client.
- `RemoteEndpoint.Async getAsyncRemote()`
- `void close()`

Interfaţa `RemoteEndpoint` are subinterfeţele
`RemoteEndpoint.Async`
`RemoteEndpoint.Basic`.

Interfaţa `javax.websocket.RemoteEndpoint.Basic`

Metode

- `void sendObject(Object` *data*`) throws IOException,EncodeException`
- `void sendText(String` *data*`) throws IOException`
- `void sendBinary(ByteBuffer` *data*`) throws IOException`
- `OutputStream getSendStream() throws IOException`

Exemplul 8.2.1 *Aplicaţie WebSocket pentru calculul unei integrale.*

Aplicaţia *server* are codul

```
1  package websocket.cvadra;
2  import javax.websocket.OnMessage;
3  import javax.websocket.server.ServerEndpoint;
4  import javax.websocket.Session;
5  import javax.websocket.OnOpen;
6  import javax.websocket.OnClose;
7  import javax.websocket.OnError;
8  import javax.websocket.RemoteEndpoint;
9  import javax.websocket.EncodeException;
10 import java.io.IOException;
11 import java.util.Set;
12 import java.util.Collections;
13 import java.util.HashSet;
14 import mathlib.client.cvadra.JepDataIn;
15 import mathlib.client.cvadra.DataOut;
16 import mathlib.client.cvadra.IMetodaSimpson;
17 import mathlib.client.cvadra.impl.MetodaSimpsonWeb;
18 import java.text.DecimalFormat;

20 @ServerEndpoint(value="/cvadra")
21 public class IntegralaWebSocketServerAd{
22   private static Set<Session> sessions =
23     Collections.synchronizedSet(new HashSet<Session>());

25   @OnMessage
26   public void onMessage(String message,Session session){
27     String[] elem=message.split(":");
28     String var=elem[0];
29     String expr=elem[1];
30     JepDataIn din=new JepDataIn(var,expr);
31     din.setA(elem[2]);
32     din.setB(elem[3]);
33     din.setEps(Double.parseDouble(elem[4]));
34     din.setNmi(Integer.parseInt(elem[5]));
35     String tip=elem[6];
36     IMetodaSimpson obj=new MetodaSimpsonWeb();
37     DataOut dout=obj.metodaSimpson(din);
38     DecimalFormat df=new DecimalFormat("0.0000001");
39     StringBuffer rez=new StringBuffer();
40     if(tip.equals("html")){
41       rez.append("<table><tr><td>");
42       rez.append("Indicatorul de raspuns : "+dout.getInd());
43       rez.append("</td></tr><tr><td>");
44       rez.append("Integrala : "+df.format(dout.getIntegrala()));
45       rez.append("</td></tr><tr><td>");
46       rez.append("Numarul iteratiilor efectuale : "+dout.getNi());
47       rez.append("</td></tr></table>");
```

```
48      }
49      else{
50        rez.append("Indicatorul de raspuns : "+dout.getInd());
51        rez.append("\n");
52        rez.append("Integrala : "+dout.getIntegrala());
53        rez.append("\n");
54        rez.append("Numarul iteratiilor efectuale : "+dout.getNi());
55      }
56      sessions.stream()
57        .filter(s->s.equals(session))
58        .forEach(s->{
59          RemoteEndpoint.Basic endpoint=s.getBasicRemote();
60          try{
61            endpoint.sendText(rez.toString());
62          }
63          catch(IOException e){};
64      });
65    }

67    @OnOpen
68    public void onOpen(Session session)
69        throws IOException,EncodeException{
70      sessions.add(session);
71    }

73    @OnClose
74    public void onClose(Session session){
75      sessions.remove(session);
76    }
77  }
```

Clientul Web HTML5/Javascript este

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
5      <title>IntegralaWebSocketAd</title>

7      <script language="javascript" type="text/javascript">
8        var wsUri = "ws://localhost:8080/IntegralaWebSocketAd/cvadra";
9        var websocket = new WebSocket(wsUri);
10       websocket.onopen = function(evt) { onOpen(evt) };
11       websocket.onmessage = function(evt) { onMessage(evt) };
12       websocket.onerror = function(evt) { onError(evt) };
13       websocket.onclose = function(evt) { onClose(evt) };

15       function init() {
16         output = document.getElementById("result");
17       }

19       function send() {
20         var svar=document.cvadra.svar.value;
21         var expr=document.cvadra.expr.value;
22         var a=document.cvadra.a.value;
23         var b=document.cvadra.b.value;
24         var eps=document.cvadra.eps.value;
25         var nmi=document.cvadra.nmi.value;
26         var msg=svar+":"+expr+":"+a+":"+b+":"+eps+":"+nmi+":html";
27         websocket.send(msg);
28         writeToScreen("SENT: " + msg);
29       }

31       function onOpen(evt) {
32         writeToScreen("CONNECTED");
33       }
```

```
35        function onMessage(evt) {
36          writeToScreen("RECEIVED: " + evt.data);
37        }

39        function onError(evt) {
40          writeToScreen('<span style="color: red;">ERROR:</span> ' + evt.data);
41        }

43        function onClose(evt) {
44          writeToScreen("CLOSED");
45        }

47        function myclose(){
48          websocket.close();
49        }

51        function writeToScreen(message) {
52          var pre = document.createElement("p");
53          pre.style.wordWrap = "break-word";
54          pre.innerHTML = message;
55          output.appendChild(pre);
56        }

58        window.addEventListener("load", init, false);
59      </script>
60    </head>
61    <body bgcolor="#bbccbb">
62      <center>
63      <h1> Pagina de apelare a aplica&#355;iei websocket  MetodaSimpson</h1>
64      <h3> Calculul unei integrale </h3>
65      <p/>Introduce&#355;i:
66      <!--<div style="text-align: center;">-->
67      <form name="cvadra">
68        <table border="2" >
69          <tr>
70            <td> Variabila independenta : </td>
71            <td> <input type="text" name="svar" size=30 required> </td>
72          </tr>
73          <tr>
74            <td> Expresia de integrat : </td>
75            <td> <input type="text" name="expr" size=30 required> </td>
76          </tr>
77          <tr>
78            <td> Limita inferioara a intervalului : </td>
79            <td> <input type="text" name="a" size=30 required> </td>
80          </tr>
81          <tr>
82            <td> Limita superioara a intervalului : </td>
83            <td> <input type="text" name="b" size=30 required> </td>
84          </tr>
85          <tr>
86            <td> Toleranta : </td>
87            <td> <input type="text" name="eps" value="1.0e-8" size=30 required> </td>
88          </tr>
89          <tr>
90            <td> Numar maxim admis de iteratii : </td>
91            <td> <input type="text" name="nmi" value="50" size=30 required> </td>
92          </tr>
93          <tr>
94            <td>
95              <input type="button" value="Calculeaza" onclick="send()"/>
96            </td>
97            <td></td>
98          </tr>
99          <tr>
100           <td>
101             <input type="button" value="Inchide conexiunea" onclick="myclose()"/>
```

```
102             </td>
103             <td></td>
104           </tr>
105         </table>
106       </form>
107       </div>
108       <div id="result"></div>
109       </center>
110     </body>
111 </html>
```

8.2.3 Client Java pentru WebSocket

Interfaţa de programare WebSocket permite realizarea de clase client. Programarea clasei utilizează în plus:

- Clasa `javax.websocket.Endpoint`

  Metode

  - `void onClose(Session` *session,* `CloseReason` *closeReason*`)`
  - `void onError(Session` *session,* `Throwable` *th*`)`
  - `abstract void onOpen(Session` *session,* `EndpointConfig` *config*`)`

- Clasa `javax.websocket.ContainerProvider`

  Metode

  - `public static WebSocketContainer getWebSocketContainer()`

- Interfaţa `javax.websocket.WebSocketContainer`

  Metode

  - `Session connectToServer(Endpoint` *endpointInstance,*
    `ClientEndpointConfig` *cec,* `URI` *path*`) throws`
    `DeploymentException,IOException`

Exemplul 8.2.2 *Program client pentru aplicaţia WebSocket IntegralaWebSocketServerAd (8.2.1).*

```
1  import java.io.IOException;
2  import java.net.URI;
3  import javax.websocket.ContainerProvider;
4  import javax.websocket.DeploymentException;
5  import javax.websocket.WebSocketContainer;
6  import javax.websocket.Session;
7  import java.util.Scanner;

9  import javax.websocket.Endpoint;
10 import javax.websocket.EndpointConfig;
11 import javax.websocket.MessageHandler;

13 public class WebSocketClient extends Endpoint{
```

```
14    private static boolean sfarsit=false;
15    private static String server = "ws://localhost:8080/IntegralaWebSocketAd/cvadra";

17    public static void main(String[] args){
18      Scanner scanner=new Scanner(System.in);
19      System.out.println("Introduceti:");
20      System.out.println("Variabila independenta ");
21      String svar=scanner.next();
22      System.out.println("Expresia de integrat ");
23      String expr=scanner.next();
24      System.out.println("Limita inferioara ");
25      String a=scanner.next();
26      System.out.println("Limita superioara ");
27      String b=scanner.next();
28      System.out.println("Toleranta ");
29      double eps=scanner.nextDouble();
30      System.out.println("Numar maxim admis de iteratii ");
31      int nmi=scanner.nextInt();
32      String msg=svar+":"+expr+":"+a+":"+b+":"+eps+":"+nmi+":text";
33      WebSocketContainer container = ContainerProvider.getWebSocketContainer();
34      try {
35        Session session=
36          container.connectToServer(WebSocketClient.class, null, URI.create(server));
37        session.getBasicRemote().sendText(msg);
38        while(! sfarsit){System.out.print("");};
39        session.close();
40      }
41      catch(Exception ex){
42         System.out.println("LocalEndPoint Exception : "+ex.getMessage());
43      }
44    }

46    public void onOpen(Session session, EndpointConfig config) {
47      session.addMessageHandler(new MessageHandler.Whole<String>() {
48          public void onMessage(String text){
49            System.out.println("Rezultate : ");
50            System.out.println(text);
51            sfarsit=true;
52          }
53      });
54    }
55  }
```

8.3 *Google Web Toolkit* (GWT)

GWT utilizează protocolul *Asynchronous JavaScript And Xml* (AJAX). La bază protocolului AJAX se află o interfaţa de programare (API) care printr-un obiect *XMLHttpRequest* (XHR), poate fi utilizată de un limbaj de scripting, în particular Javascript, pentru

- transfer de date către un server Web prin protocolul HTTP;
- manipularea datelor XML sau JSON (JavaScript Object Notation).

Caracterul asincron constă în faptul că răspunsul furnizat de un program server reface doar o parte din pagina html şi nu întrega pagină, aşa cum este cazul utilizării obişnuite a unui servlet.

Din punct de vedere al structurii aplicaţiei GWT, aceasta poate fi:

- Simplă, fără apel de procedură la distanţă. În acest caz, rezolvarea cererii este programată în clase aflate în catalogul `client`.

  Programatorul dezvoltă aplicaţia în Java şi HTML iar GWT transformă partea de client în JavaScript. Astfel se evită programarea în JavaScript.

- Cu apel de procedură la distanţă. Partea de server este bazată pe tehnologia servlet.

GWT este distribuit gratuit de firma Google.
Instalarea produsului constă din dezarhivarea fişierului descărcat din Internet.

8.3.1 Dezvoltarea unei aplicaţii cu GWT

O aplicaţie GWT o vom dezvolta cu *apache-ant*. Alternativ se poate utiliza *apache-maven* sau *eclipse.*

O aplicaţie GWT se iniţiază prin generarea unei structuri de cataloage şi fişiere. Dacă se doreşte realizarea unei aplicaţii cu punctul de intrare dat de clasa *context.MyApp* şi care să se afle într-un catalog *catapp,* atunci generarea se obţine prin comanda

`webAppCreator -out` *catapp context.MyApp*

lansată într-o fereastă DOS. *Context*ul poate reprezenta un şir de cataloage.

Rezultatul este reprezentat în Fig. 8.1 şi corespunde unei aplicaţii de întâmpinare.

```
catapp
   |--> src
   |    |--> context
   |    |    |--> client
   |    |    |    |   MyApp.java
   |    |    |    |   GreetingService.java
   |    |    |    |   GreetingServiceAsync.java
   |    |    |--> server
   |    |    |    |   GreetingServiceImpl.java
   |    |    |--> shared
   |    |    |    |   FieldVerifier.java
   |    |    |    MyApp.gwt.xml
   |--> test
   |    |    |    . . .
   |--> war
   |    |--> WEB-INF
   |    |    |    web.xml
   |    |    MyApp.css
   |    |    MyApp.html
   |    |    favicon.ico
   |    .classpath
   |    .project
   |    MyApp.launch
   |    README.txt
   |    build.xml
```

Fig. 8.1: Iniţializarea unei aplicaţii GWT.

Acestă structură reprezintă un proiect GWT. Proiectul generat este punctul de plecare

pentru construirea oricărei alte aplicaţii, a cărei dezvoltare constă în modificarea, rescrierea fişierelor create şi completarea cu altele noi. Pentru o aplicaţie GWT se mai foloseşte şi termenul de modul GWT.

Fişierul *MyApp.gwt.xml* este un fişier de configurare în care trebuie declarate modulele externe utilizate.

O aplicaţie GWT poate fi executată în

- modul de dezvoltare. Rularea în acest mod se lansează prin `ant devmode`.

  Verificarea aplicaţiei se face prin intermediul navigatorului implicit.

- modul Web, de producţie - caz în care se generează arhiva `war` a aplicaţiei. Se va executa `ant war`.

  Cu notaţiile utilizate mai sus, va rezulta fişierul *MyApp.war.* După desfăşurarea aplicaţiei într-un server Web, container de servlet, se va apela `http://host:port/MyApp/MyApp.html`.

Dezvoltarea unei aplicaţii GWT simple

După generarea proiectului, aplicaţia GWT se dezvoltă parcurgând paşii (se presupune din nou că numele aplicaţiei este *MyApp* aflat în catalogul *catapp*):

1. *Proiectarea interfeţei grafice* vizează elementele care se definesc în fişierul *MyApp.html,* punctul de intrare în aplicaţie. Un *widget*[2] (element, control) grafic va fi redat de navigator într-o fantă (slot) definită, uzual, printr-un container `div`

2. *Construirea interfeţei grafice* constă în definirea obiectelor Java care umplu fantele declarate mai sus. Acest lucru se programează în clasa *MyApp.java,* care implementează interfaţa `EntryPoint`, interfaţă ce declară doar metoda

   `public void onModuleLoad()`.

   Implementarea acestei metode reprezintă tocmai construcţia interfeţei grafice. Interfaţa de programare GWT (API) conţine o famile de clase *widget.* O instanţa a unui *widget* se asociază fantei prin

   `RootPanel.get("`*slot*`").add(`*widget*`);`

   Clasele *widget* cu metodele care vor fi utilizate sunt:

   - `Label`

     Constructori:

[2] *widget*=*gadget* virtual, *gadget*=dispozitiv amuzant, fără însemnătate practică.

  - `Label()`
  - `Label(String` *text*`)`

  Metode:

  - `public void setText(String` *text*`)`

- `TextBox`

  Constructori:

  - `TextBox()`

  Metode:

  - `public String getText()`
  - `public void setText(String` *text*`)`
  - `public void setVisibleLength(int` *lungime*`)`

- `Button`

  Constructori:

  - `Button(String` *text*`)`

  Metode:

  - `public HandlerRegistration addClickHandler(ClickHandler` *click Handler*`)`
    Metoda *clickHandler* conţine prelucrarea ataşată butonului.

- Containere de *widget*

`VerticalPanel`	`VerticalSplitPanel`
`HorizontalPanel`	`HorizontalSplitPanel`
`FlowPanel`	`DockPanel`

  Un *widget* se include într-un container cu metoda

  `void add(Widget` *widget*`)`

3. *Generarea evenimentelor.* Activităţile / acţiunile care constituie obiectivul aplicaţiei GWT se lansează printr-un clic pe un buton. Fiecărui buton i se atribuie un obiect care implementează interfaţa `ClickHandler.` Activităţile amintite mai sus sunt definite în codul metodei `public void onClick(ClickEvent` *event*`).`

4. *Programarea activităţilor corespunzătoare evenimentelor* ataşate butoanelor, adică implementarea metodelor `onClick.`

5. *Fixarea elementelor de stil* ale elementelor grafice în fişierul *MyApp.css.* Ataşarea la un *widget* a unui element de stil se obţine cu metoda `public void addStyleName(` `String` *style*`).`

Urmărim aceşti paşi în

Exemplul 8.3.1 *Calculul unei integrale. Transformăm aplicaţia 1.2 într-o aplicaţie GWT.*

Într-o aplicaţie GWT codurile Java din catalogul *client* sunt transformate în cod JavaScript. Drept consecinţă, nu se pot utiliza arhive `jar`. În locul lor, se vor folosi module GWT, care în cazul aplicaţiei de faţă sunt:

- mini-biblioteca *mathlib,* pentru calculul propriu-zis al integralei. Mini-biblioteca *mathlib* a fost organizată de la început ca modul GWT;
- *MathEclipse-Parser,* pentru evaluarea expresiilor de calcul date ca `String`.

Clasa *mathlib.client.cvadra.MEParserDataIn,* parte a mini-bibliotecii *mathlib,* extinde clasa abstractă *mathlib.client.cvadra.DataIn*:

```
1  package mathlib.client.cvadra;
2  import org.matheclipse.parser.client.eval.*;

4  public class MEParserDataIn extends DataIn{
5    private DoubleEvaluator parser=null;
6    private IDoubleValue v=null;
7    private String expr;

9    public void setA(String expr){
10     DoubleEvaluator aParser=new DoubleEvaluator();
11     super.setA(aParser.evaluate(expr));
12   }
13   public void setB(String expr){
14     DoubleEvaluator bParser=new DoubleEvaluator();
15     super.setB(bParser.evaluate(expr));
16   }

18   public MEParserDataIn(String var,String expr){
19     this.expr=expr;
20     parser=new DoubleEvaluator();
21     v=new DoubleVariable(0.0);
22     parser.defineVariable(var,v);
23   }

25   // functia de integrat
26   public double fct(double x){
27      v.setValue(x);
28      return parser.evaluate(expr);
29   }
30 }
```

Concret se vor copia unele cataloage din sursa mini-bibliotecii şi din fişierele sursă obţinute din dezarhivarea fişierului *matheclipse-parser-*.jar.*

Se generează proiectului GWT cu punctul de intrare dat de clasa `numerjava.gwt.AppIntegrala`. Structura generată de cataloage şi fişiere se completează cu resursele indicate în Fig. 8.2.

- *Proiectarea interfeţei grafice.* Considerăm interfeţa grafică

Calculul unei integrale	↔ `Label` *titleLabel*
	↔ `HorizontalPanel` *hp*
Panou container pentru *dataPanel* şi *resultsPanel*	
	↔ `VerticalPanel` *dataPanel*
Panoul stâng pentru datele furnizate de client	
	↔ `VerticalPanel` *resultsPanel*
Panou drept pentru rezultatele aplicaţiei	
Integreaza	↔ `Button` *button*

```
integrala
   |--> src
   |    |--> numerjava
   |    |    |--> gwt
   |    |    |    |--> client
   |    |    |    |    |    AppIntegrala.java
   |    |    |    |    |    . . .
   |    |    |    |    AppIntegrala.gwt.xml
   |    |--> mathlib
   |    |    |--> client
   |    |    |    |--> cvadra
   |    |    |    |    |--> impl
   |    |    |    |    |    |   MetodaSimpsonWeb.java
   |    |    |    |    |    IMetodaSimpson.java
   |    |    |    |    |    DataIn.java
   |    |    |    |    |    DataOut.java
   |    |    |    |    |    MEParserDataIn.java
   |    |    |    |    Mathlib.gwt.xml
   |    |--> org
   |    |    |--> matheclipse
   |    |    |    |--> parser
   |    |    |    |    |    . . .
   |    |    . . .
   |    |--> war
   |    |    |--> WEB-INF
   |    |    |    |    . . .
   |    |    |    MyApp.css
   |    |    |    MyApp.html
   |    . . .
```

Fig. 8.2: Iniţializarea unei aplicaţii GWT.

unde panourile *dataPanel* şi *resultsPanel* conţin

Simbolul variabilei	↔	`Label` *varLabel*
▭	↔	`TextBox` *varTextBox*
pentru introducerea simbolului variabilei de integrare		
Functia	↔	`Label` *exprLabel*
▭	↔	`TextBox` *exprTextBox*
pentru introducerea expresiei de integrat		
Limita inferioara	↔	`Label` *infLabel*
▭	↔	`TextBox` *infTextBox*
pentru introducerea limitei inferioara a intervalului		
Limita superioara	↔	`Label` *supLabel*
▭	↔	`TextBox` *supTextBox*
pentru introducerea limitei superioare a intervalului		
Toleranta	↔	`Label` *epsLabel*
▭	↔	`TextBox` *epsTextBox*
pentru introducerea toleranţei		
Numar maxim admis de iteratii	↔	`Label` *nmiLabel*
▭	↔	`TextBox` *nmiTextBox*
pentru introducerea numărului maxim admis de iteraţii		

şi respectiv

Indicatorul de raspuns	↔	`Label` *indLabel*
Integrala	↔	`Label` *integLabel*
Numarul de iteratii efectuat	↔	`Label` *niLabel*

*Widget*ele *titleLabel, hp, button* vor fi redate în fantele declarate respectiv prin

```
<body class="bd">
  <center>
    <div id="titleLabel"> </div>
    <p><div id="mainPanel"></div></p>
    <p><div id="button"></div></p>
  </center>
</body>
```

- *Construirea interfeţei grafice.* Codul care implementează interfaţa grafică imaginată mai sus este

```
public void onModuleLoad() {
    Label titleLabel=new Label("Calculul unei integrale");
    Label varLabel=new Label("Simbolul variabilei");
    Label exprLabel=new Label("Functia");
    Label infLabel=new Label("Limita inferiara");
    Label supLabel=new Label("Limita superioara");
    Label epsLabel=new Label("Toleranta");
    Label nmiLabel=new Label("Numar maxim admis de iteratii");
    TextBox varTextBox=new TextBox();
    varTextBox.setVisibleLength(20);
    TextBox exprTextBox=new TextBox();
    exprTextBox.setVisibleLength(20);
    TextBox infTextBox=new TextBox();
    infTextBox.setVisibleLength(20);
    TextBox supTextBox=new TextBox();
    supTextBox.setVisibleLength(20);
    TextBox epsTextBox=new TextBox();
    epsTextBox.setVisibleLength(20);
    epsTextBox.setText("1e-5");
    TextBox nmiTextBox=new TextBox();
    nmiTextBox.setText("50");
    nmiTextBox.setVisibleLength(20);

    VerticalPanel dataPanel=new VerticalPanel();
    dataPanel.add(varLabel);    dataPanel.add(varTextBox);
    dataPanel.add(exprLabel);   dataPanel.add(exprTextBox);
    dataPanel.add(infLabel);    dataPanel.add(infTextBox);
    dataPanel.add(supLabel);    dataPanel.add(supTextBox);
    dataPanel.add(epsLabel);    dataPanel.add(epsTextBox);
    dataPanel.add(nmiLabel);    dataPanel.add(nmiTextBox);
    dataPanel.setBorderWidth(2);

    Label indLabel=new Label("Indicatorul de raspuns");
    Label integLabel=new Label("Integrala");
    Label niLabel=new Label("Numarul de iteratii efectuat");

    VerticalPanel resultsPanel=new VerticalPanel();
    resultsPanel.add(indLabel);
    resultsPanel.add(integLabel);
    resultsPanel.add(niLabel);
    resultsPanel.setBorderWidth(2);

    HorizontalPanel hp=new HorizontalPanel();
    hp.setSpacing(10);
    hp.add(dataPanel);
    hp.add(resultsPanel);
```

```
    Button button=new Button("Integreaza");
    RootPanel.get("titleLabel").add(titleLabel);
    RootPanel.get("mainPanel").add(hp);
    RootPanel.get("button").add(button);
  }
}
```

- *Generarea evenimentelor.* Butonului i se asociază o instanţă a clasei *MyClickHandler,* care conţine acţiunile executate după clic pe buton.

```
MyClickHandler handler=new MyClickHandler(varTextBox,
      exprTextBox,infTextBox,supTextBox,epsTextBox,nmiTextBox,
      indLabel,integLabel,niLabel);
button.addClickHandler(handler);
```

- *Programarea activităţilor corespunzătoare evenimentelor.* Acest pas corespunde realizării clasei *MyClickHandler.* Acţiunile care se execută constau din verificarea completării fiecărui câmp al formularului, urmată de calculul integralei.

  Codul clasei *MyClickHandler* este

```
1  class MyClickHandler implements ClickHandler{
2    TextBox varTextBox;
3    TextBox exprTextBox;
4    TextBox infTextBox;
5    TextBox supTextBox;
6    TextBox epsTextBox;
7    TextBox nmiTextBox;
8    Label indLabel;
9    Label integLabel;
10   Label niLabel;

12   MyClickHandler(TextBox varTextBox,TextBox exprTextBox,
13     TextBox infTextBox,TextBox supTextBox,
14     TextBox epsTextBox,TextBox nmiTextBox,
15     Label indLabel,Label integLabel,Label niLabel){
16     this.varTextBox=varTextBox;
17     this.exprTextBox=exprTextBox;
18     this.infTextBox=infTextBox;
19     this.supTextBox=supTextBox;
20     this.epsTextBox=epsTextBox;
21     this.nmiTextBox=nmiTextBox;
22     this.indLabel=indLabel;
23     this.integLabel=integLabel;
24     this.niLabel=niLabel;
25   }

27   public boolean isCompleted(TextBox tb,String name){
28     String txt=tb.getText();
29     if(txt.equals("")){
30       Window.alert("Camp necompletat : "+name);
31       indLabel.setText("?");
32       integLabel.setText("?");
33       niLabel.setText("?");
34       return false;
35     }
36     return true;
37   }

39   public void onClick(ClickEvent event){
40     if(!isCompleted(varTextBox,"Simbolul variabilei")) return;
41     if(!isCompleted(exprTextBox,"Functia")) return;
42     if(!isCompleted(infTextBox,"Limita inferioara")) return;
43     if(!isCompleted(supTextBox,"Limita superioara")) return;
```

```
44      if(!isCompleted(epsTextBox,"Toleranta")) return;
45      if(!isCompleted(nmiTextBox,"Numar maxim admin de iteratii")) return;
46      MEParserDataIn din=
47        new MEParserDataIn(varTextBox.getText(),exprTextBox.getText());
48      din.setA(infTextBox.getText());
49      din.setB(supTextBox.getText());
50      String eps=epsTextBox.getText();
51      String nmi=nmiTextBox.getText();
52      din.setEps(Double.parseDouble(eps));
53      din.setNmi(Integer.parseInt(nmi));
54      IMetodaSimpson obj=new MetodaSimpsonWeb();
55      DataOut dout=obj.metodaSimpson(din);
56      indLabel.setText("Indicatorul de raspuns : "+dout.getInd());
57      integLabel.setText("Integrala : "+
58        NumberFormat.getFormat("###0.000000").format(dout.getIntegrala()));
59      niLabel.setText("Numarul iteratiilor efectuate : "+dout.getNi());
60    }
61  }
```

- *Fixarea elementelor de stil.* Fişierul *AppIntegrala.css* conţine

```
.button {
  display: block;
  font-size: 18pt;
  color: blue
}

.label-title {
  font-weight: bold;
  font-size: 20pt;
  color: blue
}

.bd{
    background-color: #bbccbb;
}
```

Codul complet al clasei *AppIntegrala.java* este

```
1  package numerjava.gwt.client;

3  import com.google.gwt.core.client.EntryPoint;
4  import com.google.gwt.user.client.ui.*;
5  import com.google.gwt.user.client.Window;
6  import com.google.gwt.i18n.client.NumberFormat;
7  import com.google.gwt.event.dom.client.ClickEvent;
8  import com.google.gwt.event.dom.client.ClickHandler;
9  import mathlib.client.cvadra.*;
10 import mathlib.client.cvadra.impl.MetodaSimpsonWeb;

12 public class AppIntegrala implements EntryPoint {

14   public void onModuleLoad() {
15     Label titleLabel=new Label("Calculul unei integrale");
16     titleLabel.addStyleName("label-title");
17     Label varLabel=new Label("Simbolul variabilei");
18     Label exprLabel=new Label("Functia");
19     Label infLabel=new Label("Limita inferiara");
20     Label supLabel=new Label("Limita superioara");
21     Label epsLabel=new Label("Toleranta");
22     Label nmiLabel=new Label("Numar maxim admis de iteratii");
23     TextBox varTextBox=new TextBox();
24     varTextBox.setVisibleLength(20);
25     TextBox exprTextBox=new TextBox();
26     exprTextBox.setVisibleLength(20);
```

```
27      TextBox infTextBox=new TextBox();
28      infTextBox.setVisibleLength(20);
29      TextBox supTextBox=new TextBox();
30      supTextBox.setVisibleLength(20);
31      TextBox epsTextBox=new TextBox();
32      epsTextBox.setVisibleLength(20);
33      epsTextBox.setText("1e-5");
34      TextBox nmiTextBox=new TextBox();
35      nmiTextBox.setText("50");
36      nmiTextBox.setVisibleLength(20);

38      VerticalPanel dataPanel=new VerticalPanel();
39      dataPanel.add(varLabel);     dataPanel.add(varTextBox);
40      dataPanel.add(exprLabel);    dataPanel.add(exprTextBox);
41      dataPanel.add(infLabel);     dataPanel.add(infTextBox);
42      dataPanel.add(supLabel);     dataPanel.add(supTextBox);
43      dataPanel.add(epsLabel);     dataPanel.add(epsTextBox);
44      dataPanel.add(nmiLabel);     dataPanel.add(nmiTextBox);
45      dataPanel.setBorderWidth(2);

47      Label indLabel=new Label("Indicatorul de raspuns");
48      Label integLabel=new Label("Integrala");
49      Label niLabel=new Label("Numarul de iteratii efectuat");

51      VerticalPanel resultsPanel=new VerticalPanel();
52      resultsPanel.add(indLabel);
53      resultsPanel.add(integLabel);
54      resultsPanel.add(niLabel);
55      resultsPanel.setBorderWidth(2);

57      HorizontalPanel hp=new HorizontalPanel();
58      hp.setSpacing(10);
59      hp.add(dataPanel);
60      hp.add(resultsPanel);

62      Button button=new Button("Integreaza");
63      button.addStyleName("button");
64      MyClickHandler handler=new MyClickHandler(varTextBox,
65        exprTextBox,infTextBox,supTextBox,epsTextBox,nmiTextBox,
66        indLabel,integLabel,niLabel);
67      button.addClickHandler(handler);

69      RootPanel.get("titleLabel").add(titleLabel);
70      RootPanel.get("mainPanel").add(hp);
71      RootPanel.get("button").add(button);
72    }
73  }

75  class MyClickHandler implements ClickHandler{
76    TextBox varTextBox;
77    TextBox exprTextBox;
78    TextBox infTextBox;
79    TextBox supTextBox;
80    TextBox epsTextBox;
81    TextBox nmiTextBox;
82    Label indLabel;
83    Label integLabel;
84    Label niLabel;

86    MyClickHandler(TextBox varTextBox,TextBox exprTextBox,TextBox infTextBox,
87      TextBox supTextBox,TextBox epsTextBox,TextBox nmiTextBox,
88      Label indLabel,Label integLabel,Label niLabel){
89      this.varTextBox=varTextBox;
90      this.exprTextBox=exprTextBox;
91      this.infTextBox=infTextBox;
92      this.supTextBox=supTextBox;
93      this.epsTextBox=epsTextBox;
```

```
94      this.nmiTextBox=nmiTextBox;
95      this.indLabel=indLabel;
96      this.integLabel=integLabel;
97      this.niLabel=niLabel;
98    }

100   public boolean isCompleted(TextBox tb,String name){
101     String txt=tb.getText();
102     if(txt.equals("")){
103       Window.alert("Camp necompletat : "+name);
104       indLabel.setText("?");
105       integLabel.setText("?");
106       niLabel.setText("?");
107       return false;
108     }
109     return true;
110   }

112   public void onClick(ClickEvent event){
113     if(!isCompleted(varTextBox,"Simbolul variabilei")) return;
114     if(!isCompleted(exprTextBox,"Functia")) return;
115     if(!isCompleted(infTextBox,"Limita inferioara")) return;
116     if(!isCompleted(supTextBox,"Limita superioara")) return;
117     if(!isCompleted(epsTextBox,"Toleranta")) return;
118     if(!isCompleted(nmiTextBox,"Numar maxim admin de iteratii")) return;
119     MEParserDataIn din=new MEParserDataIn(varTextBox.getText(),exprTextBox.getText());
120     din.setA(infTextBox.getText());
121     din.setB(supTextBox.getText());
122     String eps=epsTextBox.getText();
123     String nmi=nmiTextBox.getText();
124     din.setEps(Double.parseDouble(eps));
125     din.setNmi(Integer.parseInt(nmi));
126     IMetodaSimpson obj=new MetodaSimpsonWeb();
127     DataOut dout=obj.metodaSimpson(din);
128     indLabel.setText("Indicatorul de raspuns : "+dout.getInd());
129     integLabel.setText("Integrala : "+
130       NumberFormat.getFormat("###0.000000").format(dout.getIntegrala()));
131     niLabel.setText("Numarul iteratiilor efectuate : "+dout.getNi());
132   }
133 }
```

Conţinutul fişierului de configurare al modulului *mathlib* (*Mathlib.gwt.xml*) este

```
<module>
    <inherits name="com.google.gwt.user.User"/>
    <inherits name="org.matheclipse.parser.Parser"/>
 </module>
```

iar fişierul *AppIntegrala.gwt.xml* se completează cu

```
  <inherits name="org.matheclipse.parser.Parser"/>
  <inherits name="mathlib.Mathlib"/>
```

Interfaţa grafică a aplicaţiei este prezentată în Fig. 8.3.

8.4 Desfăşurarea în *nor*

Dezvoltarea Internetului, nevoia de a reduce costurile legate de realizarea şi întreţinerea infrastructurii care oferă servicii pe Internet, concomitent cu nevoia de creştere a calităţii serviciilor a condus la *servicii în nor* (*Cloud Computing*).

Avantajele oferite de serviciile *serviciile în nor* sunt:

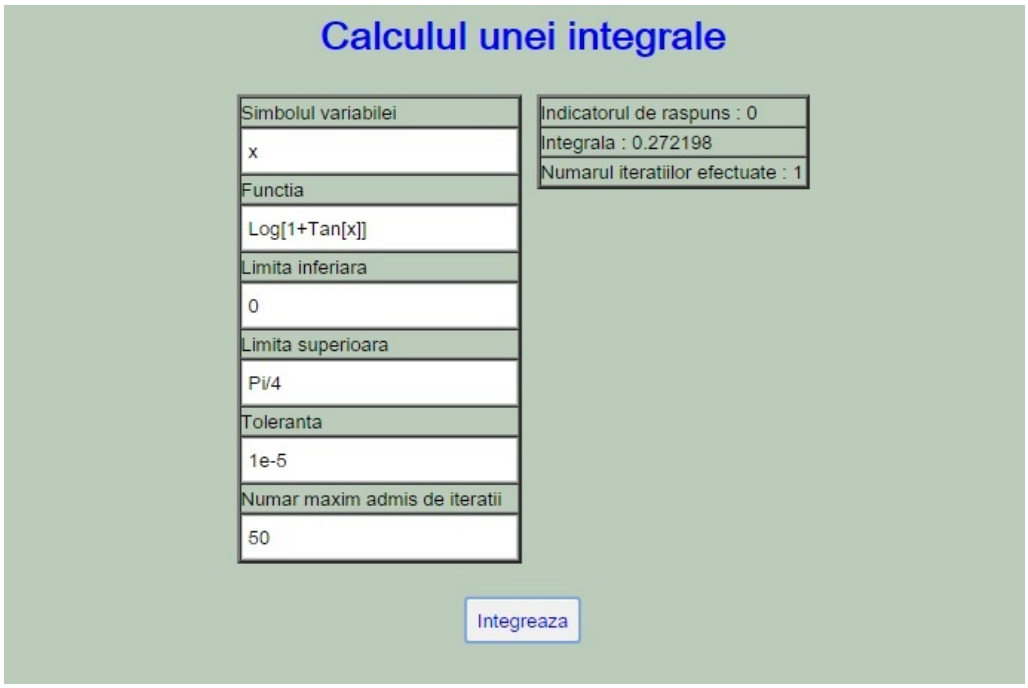


Fig. 8.3: Aplicaţia GWT *AppIntegrala*.

- Reducerea costurilor
- Agilitate (*Agility*)

  Reducerea duratei:

  – de aşteptare în cazul apariţiei unei disfuncţionalităţi din partea furnizorului *serviciului în nor;*

  – de actualizare şi întreţinere din partea realizatorului *serviciului în nor.*

- Elasticitate (*Elasticity*)

  Posibilitatea de creştere / descreştere a resurselor (în principal hard) alocate pentru a satisface cerinţele clienţilor într-un interval de timp.

  Se face distincţie de *scalabilitate,* termen care desemnează nevoia de creşere / descreştere a resurselor alocate legată de dezvoltarea aplicaţiilor care compun serviciul.

Tipuri de *servicii în nor:*

- Aplicaţii ca serviciu (*Software as a Service - SaaS*)

  Skype, Google's Docs, Gmail, Yahoo Messenger, Microsoft Office 365, etc.

- Infrastructură ca serviciu (*Infrastructure as a Service - IaaS*)

  Amazon's Elastic Compute Cloud - (EC2)

- Plaformă ca serviciu (*Platform as a Service - PaaS*)

  PaaS poate fi

  – Ne-portablă : aplicaţia va avea o structură predefinită.
  Google AppEngine (GAE), *Microsoft Azure*, *OpenShift*

  – Portabilă
  Heroku

În cele ce urmează ne interesează doar platformele PaaS care acceptă desfăşurarea de aplicaţii Java, în mod gratuit (cel putin pentru un număr redus de aplicaţii), sau oferă un simulator local. Serviciile în nor *Google AppEngine, Heroku, OpenShift* asigură aceste condiţii. Modul de tratare este specifică fiecărui serviciu în nor în parte.

Vom exemplifica doar utilizarea simulatorului pentru *Google App Engine* (GAE) care permite

- încărcarea unei aplicaţii Web pe un simulator local al platformei de Cloud Computing;

- încărcarea unei aplicaţii Web pe platforma Google de Cloud Computing.

În prezent, pe platforma GAE se pot încărca aplicaţii realizate în Java, Python, PHP şi Go, alături de care care pot apărea fişiere *http, css, js.* Există câte o distribuţie distinctă pentru fiecare din aceste limbaje de programare.

Versiunea GAE pentru Java necesită Java 7 şi *servlet-api 2.5.*

Utilizarea simulatorului local

Distribuţia GAE pentru Java conţine şablonul unei aplicaţii (`appengine-java-sdk-*\demos\new_project_template`) împreună cu un fişier `build.xml` prin intermediul căruia, cu ajutorul lui *apache-ant,* se construieşte aplicaţia în vederea verificării pe simulator / încărcării în nor.

Programatorul va înlocui aplicaţia din şablon cu propria aplicaţie, eventual va face adaptările necesare şi va rula *apache-ant.*

Lansarea simulatorului se poate face prin comenzile

```
set GAE\_HOME=. . .\appengine-java-sdk-*
%GAE\_HOME%\bin\dev_appserver war
```

lansate într-o fereastă DOS, în catalogul care conţine catalogul *war.* Aplicaţia se apelează prin `http://localhost:8080.` Dacă în loc de *index.html* se utilizează alt nume, atunci apelarea aplicaţiei este
`http://localhost:8080/fişier.html.`

Exemplul 8.4.1 *Integrarea servlet-ului MetodaSimpsonServet, dezvoltată într-o secţiune anterioară, în platforma Google App Engine.*

Şablonul aplicaţiei se copiază într-o zonă de lucru sub numele *appintegrala* şi se completează cu fişierele servlet-ului (*MetodaSimpsonServlet.java, index.html*) şi catalogul *lib.* Rezultatul va fi

```
appintegrala
|--> src
|    |--> integrala
|    |    |   MetodaSimpsonServlet.java
|    |--> META-INF
|    |    |   . . .
|    |    log4j.properties
|    |    logging.properties
|    |--> WEB-INF
|    |    |--> lib
|    |    |    |   jep-2.4.1.jar
|    |    |    |   mathlib.jar
|    |    |    appengine-web.xml
|    |    |    web.xml
|    |    index.html
|    build.xml
```

Singurul fişier specific GAE este *appengine-web.xml,* dar adaptarea acestuia se face doar pentru încărcarea în nor.

Capitolul 9

Încărcarea unui fişier - upload

Problema pe care o tratăm constă în transferul unui volum mare de date ale clientului către aplicaţia server. În particular, considerăm cazul în care datele corespund unei matrice.

La început vom distinge cazul în care matricea are dimensiuni rezonabile, putând fi introdusă într-o pagină `html`. În acest scop se vor utiliza funcţii *Javascript.*

În cazul în care volumul datelor de preluat de către aplicaţia Web este mare, calea de urmat constă din:

1. Scrierea / depozitarea datelor într-un fişier;
2. Încărcarea fişierului în aplicaţia Web.

Expedierea datelor dintr-un fişier şi recepţionarea lor defineşte problema încărcării unui fişier (*file upload*). Un produs care ne ajută să îndeplinim acest obiectiv este pachetul *commons-fileupload* - dezvoltat de *apache.*

9.1 Preluarea unei matrice prin funcţii *Javascript*

Preluarea elementelor unei matrice sau vector de dimensiune redusă, necesare unei aplicaţii Web, se poate programa în mod elegant utilizând funcţii *Javascript.* Exemplificăm prin

Exemplul 9.1.1 *Preluarea şi transmiterea unei matrice la un servlet.*

Pentru început se preiau într-un formular `html` numărul liniilor şi ale coloanelor.

```
1  <!doctype html>
2    <head>
3      <title> MatrixEnterForm </title>
4      <script type="text/javascript">
5      <!--
6      function initRequest() {
7        if (window.XMLHttpRequest) {
8            return new XMLHttpRequest();
9        } else if (window.ActiveXObject){
10           return new ActiveXObject("Microsoft.XMLHTTP");
11       }
```

```
12      }

14      function compute() {
15        var mField=document.getElementById("rows");
16        var nField=document.getElementById("cols");
17        var url = "/jsmatrix/matrix?m=" +
18            escape(mField.value)+"&n="+escape(nField.value)+"&mat="+escape(array2String());
19        var req = initRequest();
20        req.onreadystatechange = function() {
21            if (req.readyState == 4) {
22                if (req.status == 200) {
23                    parseMessages(req.responseXML);
24                } else {
25                    alert(req.status+" : "+req.statusText);
26                }
27            }
28        };
29        req.open("get", url, true);
30        req.send(null);
31      }

33      function parseMessages(responseXML) {
34        var r = responseXML.getElementsByTagName("rezultat")[0];
35        document.getElementById("rezultat").innerHTML="";
36        for (i=0;i<r.childNodes.length;i++){
37          document.getElementById("rezultat").innerHTML=
38            document.getElementById("rezultat").innerHTML+"<br/>";
39          var row=r.childNodes[i];
40          for (j=0;j<row.childNodes.length;j++){
41              var col=row.getElementsByTagName("col")[j];
42              var t=col.childNodes[0].nodeValue;
43              document.getElementById("rezultat").innerHTML=
44                document.getElementById("rezultat").innerHTML+t+"  ";
45          }
46        }
47      }

49      function templateMatrix(){
50        var rows=document.myForm.rows.value;
51        var cols=document.myForm.cols.value;
52        var myInput="";
53        document.getElementById("matrix").innerHTML="";
54        for(var i=0;i<rows;i++){
55          myInput="<br/>";
56          document.getElementById("matrix").innerHTML=
57            document.getElementById("matrix").innerHTML+myInput;
58          for(var j=0;j<cols;j++){
59            myInput="<input type='number' step='any' name='mat_"+i+
60              "_"+j+"' size='5' id='mat_"+i+"_"+j+"'/>";
61            document.getElementById("matrix").innerHTML=
62              document.getElementById("matrix").innerHTML+myInput;
63          }
64        }
65      }

67      function array2String(){
68        var rows=document.myForm.rows.value;
69        var cols=document.myForm.cols.value;
70        var x=new Array(rows);
71        for(var i=0;i<rows;i++){
72          x[i]=new Array(cols)
73          for(var j=0;j<cols;j++){
74            x[i][j]=parseFloat(document.getElementById("mat_"+i+"_"+j).value);
75          }
76        }
77        return x.toString();
78      }
```

```
79      //-->
80      </script>
81    </head>
82    <body>
83      <h1> Matrix Enter Form </h1>
84      <br/>
85      <form name="myForm">
86        <table>
87          <tr>
88            <td> Rows </td>
89            <td>
90              <input type="number" id="rows" name="rows" size="10" required min="1" />
91            </td>
92          </tr>
93          <tr>
94            <td> Columns </td>
95            <td>
96              <input type="number" id="cols" name="cols" size="10" required min="1" />
97            </td>
98          </tr>
99          <tr>
100           <td></td>
101           <td>
102             <input type="button" onClick="templateMatrix()"
103                    name="matrix" value="Generate array" />
104           </td>
105         </tr>
106       </table>
107                 <div id="matrix"> </div>
108       <p/>
109       <input type="button" value="Calculeaza" onClick="compute()" >
110     </form>
111     <div id="rezultat"></id>
112   </body>
113 </html>
```

Containerul care va conţine şablonul matricei este

```
<div id="matrix"> </div>
```

Funcţia *Javascript templateMatrix* înscrie în container şablonul matricei, adică câte un câmp de introducere date pentru fiecare element al matricei. Generarea şablonului are loc la acţionarea butonului *Generate array*.

Pe partea de server, preluarea datelor se realizează printr-o aplicaţie de tip servlet a cărei singură acţiune este afişarea datelor recepţionate.

Transmisia dintre clientul Web şi server se realizează prin tehnologia *Asynchronous JavaScript And Xml* (AJAX).

Codul servlet-ului este

```
1  package matrix;
2  import java.io.*;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import javax.servlet.annotation.WebServlet;

7  @WebServlet(urlPatterns = "/matrix")
8  public final class ReadMatrixServlet extends HttpServlet {
9    public void doGet(HttpServletRequest req,HttpServletResponse res)
10       throws ServletException,IOException{
11     PrintWriter out=res.getWriter();

13     String rows=req.getParameter("m");
14     int m=Integer.parseInt(rows);
```

```
15      String cols=req.getParameter("n");
16      int n=Integer.parseInt(cols);
17      String mat=req.getParameter("mat");
18      double[][] t=new double[m][n];
19      String[] s=mat.split(",");
20      for(int i=0;i<m;i++)
21        for(int j=0;j<n;j++)
22          t[i][j]=Double.parseDouble(s[i*n+j]);
23      res.setContentType("text/xml");
24        res.setHeader("Cache-Control","no-cache");
25        out.print("<?xml version=\"1.0\" ?>");
26        out.print("<rezultat>");
27        for(int i=0;i<m;i++){
28          out.print("<row>");
29            for(int j=0;j<n;j++){
30              out.print("<col>");
31                out.print(new Double(t[i][j]).toString());
32              out.print("</col>");
33            }
34          out.print("</row>");
35        }
36        out.print("</rezultat>");
37      out.close();
38    }

40    public void doPost(HttpServletRequest req,HttpServletResponse res)
41       throws ServletException,IOException{
42      doGet(req,res);
43    }
44  }
```

9.2 FileUpload

Transferarea unui fişier, din partea clientului nu ridică nicio problemă. În fişierul html de apelare (client Web) se defineşte un formular

```
<form
   action=. . .
   enctype="multipart/form-data"
   method="post">
```

iar un fişier de încărcat se fixează prin intermediul marcajului

```
<input type="file" name=. . . size=. . .>
```

Programul navigator afişează o fereastră de căutare, prin care clientul selectează fişierul pe care doreşte să-l încarce.

```
1  <!doctype html>
2  <body>
3  <center>
4    <h1> Rezolvarea unui sistem algebric de ecua&#355;ii liniare </h1>
5    <form
6      action="/linear/linear"
7      enctype="multipart/form-data"
8      method="post">
9      <h3> Introduce&#355;i fi&#351;ierul cu datele sistemului </h3>
10     <table>
```

```
11        <tr>
12          <td> <input type="file" name="myfile" size=30 required> </td>
13        </tr>
14        <tr>
15          <td> <input type="submit" value="Expediaza"> </td>
16        </tr>
17      </table>
18      <input type="hidden" name="tip" value="text/html" >
19    </form>
20  </center>
21  </body>
22  </html>
```

Dacă partea de client este un program, atunci se utilizează *commons-httpclient,* prezentat în 8.1.2.

Pe partea serverului utilizăm produsul *apache Commons-FileUpload* care simplifică transferul unui fişier de la un client la un servlet.

Din distribuţia produsului fişierele *commons-fileupload-*.jar, commons-io-*.jar* se depun în catalogul `WEB-INF\lib` al servlet-ului.

Programarea încărcării revine la:

1. Declararea pachetelor din *commons-fileupload* utilizate

   ```
   import org.apache.commons.fileupload.disk.*;
   import org.apache.commons.fileupload.servlet.*;
   import org.apache.commons.fileupload.*;
   ```

2. Crearea unei *fabrici* pentru manipularea fişierelor pe disc

   ```
   FileItemFactory factory = new DiskFileItemFactory();
   ```

3. Crearea unei unelte de încărcare

   ```
   ServletFileUpload   upload = new ServletFileUpload(factory);
   ```

4. Analiza (parsarea) mesajului furnizat de client

   ```
   List fileItems = upload.parseRequest(req);
   ```

   Fiecare element al listei implementează interfaţa `FileItem`.

   Se pot fixa parametrii:

   - dimensiunea zonei de pe disc destinată datelor de încărcat

     ```
     DiskFileItemFactory factory = new DiskFileItemFactory();
     factory.setSizeThreshold(maxMemorySize);
     ```

   - catalogul temporar de reţinere a datelor de încărcat

     ```
     factory.setRepositoryPath(tempDirectory);
     ```

sau direct

```
DiskFileItemFactory factory = new DiskFileItemFactory(
  maxMemorySize, tempDirectory);
```

- dimensiunea maximă a unui fişier

```
upload.setSizeMax(maxRequestSize);
```

5. Prelucrarea elementelor încărcate

```
Iterator iter=fileItems.iterator();
while (iter.hasNext()) {
  FileItem item = (FileItem) iter.next();
  if (item.isFormField()) {
    // Prelucrarea elementului item care corespunde unei
    // date din formularul html care nu este de tip fisier
  }
  else{
    // Prelucrarea elementului item de tip fisier
  }
}
```

6. În cazul unui element care nu este de tip fişier putem obţine numele şi valoarea atributului furnizat de client

```
String name = item.getFieldName();
String value = item.getString();
```

7. În cazul unui fişier putem afla numele câmpului `input` cu `type="file"`, numele fişierului, dimensiunea fişierului

```
String fieldName = item.getFieldName();
String fileName = item.getName();
long sizeInBytes = item.getSize();
```

8. Dacă dorim să salvăm fişierul pe calculatorul server atunci prelucrarea este

```
File uploadedFile = new File(...);
item.write(uploadedFile);
```

9. Dacă datele fişierului se încarcă în memoria calculatorului atunci prelucrarea este

```
InputStream in = item.getInputStream();
  //preluarea datelor din fluxul in
  . . .
in.close();
```

Alternativ, datele se pot reţine ca un şir de octeţi prin

```
byte[] data = item.get();
```

Exemplul 9.2.1 *Pentru rezolvarea sistemului algebric de ecuaţii liniare* $Ax = b$, $A \in M_{m,n}(\mathbb{R}), b \in \mathbb{R}^m$ *se formează matricea* $[A, b]$ *ale cărei elemente se reţin într-un fişier text astfel încât fiecare linie conţine o linie a matricei, iar elementele sunt separate prin spaţii. Matricea se încarcă într-un servlet unde se va utiliza clasa RezolvitorScilab din mini-biblioteca mathlib pentru rezolvarea sistemului.*

În servlet, preluarea matricei se face apelând metodele unui obiect *mathlib.client.linear.DataIn* introdus în 2.3. Servletul are codul

```
1  package linear;
2  import java.io.*;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5  import java.util.*;
6  import org.apache.commons.fileupload.disk.*;
7  import org.apache.commons.fileupload.servlet.*;
8  import org.apache.commons.fileupload.*;
9  import mathlib.client.linear.*;
10 import mathlib.client.linear.impl.RezolvitorScilab;
11 import java.text.DecimalFormat;

13 public class SistemLiniarServlet extends HttpServlet{
14   public void doPost(HttpServletRequest req,HttpServletResponse res)
15   throws ServletException,IOException {
16     String tip="";
17     ServletOutputStream out=null;
18     DecimalFormat f=new DecimalFormat("0.0000E0");
19     try{
20       FileItemFactory factory = new DiskFileItemFactory();
21       ServletFileUpload    upload = new ServletFileUpload(factory);
22       List items = upload.parseRequest(req);
23       upload.setSizeMax(1000000);
24       Iterator iter=items.iterator();
25       DataIn din=new DataIn();
26       while (iter.hasNext()) {
27         FileItem item = (FileItem) iter.next();
28         if (!item.isFormField()) {
29           InputStream in=item.getInputStream();
30           InputStreamReader isr=new InputStreamReader(in);
31           BufferedReader br=new BufferedReader(isr);
32           din.setMatrix(br);
33           br.close();
34           isr.close();
35           in.close();
36         }
37         else{
38           String name = item.getFieldName();
39           if(name.equals("tip"))
40             tip=item.getString();
41         }
42       }
43       res.setContentType(tip);
44       out = res.getOutputStream();
45       IRezolvitorScilab obj=new RezolvitorScilab();
46       DataOut dout=obj.rezolvitorScilab(din);
47       double[]x=null;
48       double[][]k=null;
49       int l=0,c=0;
```

```
50        if(dout.getECompatibil()){
51          x=dout.getX();
52          k=dout.getK();
53          l=x.length;
54          if(k!=null) c=k[0].length;
55        }
56        if(tip.equals("text/html")){
57          out.println("<html><body bgcolor=\"#bbccbb\" >");
58          out.println("<center>");
59          out.println("<h1> Solu&#355;ia sistemului algebric"+
60            " de ecua&#355;ii liniare</h1>");
61          if(dout.getECompatibil()){
62            out.println("<table border=1 cellspacing=5>");
63            out.println("<tr>");
64            out.println("<th> X= </th>");
65            out.println("<th align=\"center\" colspan="+(c-1)+"> Ker= </th>");
66            out.println("</tr>");
67            for(int i=0;i<l;i++){
68              out.println("<tr>");
69              out.println("<td>");
70              out.println(f.format(x[i]));
71              out.println("</td>");
72              if(c>0){
73                for(int j=0;j<c;j++){
74                  out.println("<td>");
75                  out.println(f.format(k[i][j]));
76                  out.println("</td>");
77                }
78              }
79              out.println("</tr>");
80            }
81            out.println("</table>");
82          }
83          else{
84            out.println("Sistem incompatibil !");
85          }
86          out.println("</center>");
87          out.println("</html></body>");
88        }
89        else{
90          String rez="";
91          if(dout.getECompatibil()){
92            for(int i=0;i<l;i++){
93              rez+=x[i];
94              rez+=" ";
95              if(c>0){
96                for(int j=0;j<c;j++){
97                  rez+=k[i][j];
98                  rez+=" ";
99                }
100             }
101             rez+='\n';
102           }
103         }
104         System.out.println(rez);
105         out.println(rez.trim());
106       }
107     }
108     catch(Exception e){
109       System.out.println("Exception: "+e.getMessage());
110     }
111     out.close();
112   }
113 }
```

Capitolul 10

Servicii Web

Un serviciu Web este o aplicaţie client-server cu serverul găzduit de un server Web, apelabil prin aplicaţia client şi realizat potrivit unei interfeţe de programare specifice. Protocolul de comunicaţie este `http`.

Sunt cunoscute următoarele tipuri de servicii Web:

- Servicii bazate pe modelul Remote Precedure Call (RPC) - Apel de Procedură de la Distanţă.

  Protocolul de reprezentare a cererii şi a răspunsului (de serializare / deserializare) variază. Din acest punct de vedere sunt cunoscute:

  - Servicii *xml-rpc* (www.xmlrpc.org). Cererea şi răspunsul sunt transmise prin cod `xml` cuprins în corpul mesajului `http`.
  - Servicii *json-rpc* (www.json-rpc.org) bazat pe reprezentarea JSON.
  - Servicii *hessian.* Se utilizează un protocol pentru serializare / deserealizare bazat pe reprezentarea binară a datelor. Protocolul a fost dezvoltat de firma *Caucho Technologies* (2007).
  - Servicii bazate pe interfaţa de programare *Java API for XML Web Services - JAX-WS.* Interfaţa de programare JAX-WS este varianta cea mai recentă pentru serviciile cunoscute sub numele de servicii *soap-rpc.*

  Pentru fiecare caz semnalat mai sus sunt realizate implementări în mai multe limbaje / platforme de programare.

- Servicii REST.

  REpresentational State Transfer (REST) este un model de arhitectură de aplicaţie distribuită[1].

  REST specifică modul cum o resursă - entitate care conţine informaţie specifică - este definită şi cum poate fi adresată.

[1]REST a fost introdus de Roy Fielding, în teza sa de doctorat din 2000. Roy Fielding este autorul principal al specificaţiilor protocolului http.

Identificarea unei resurse se face printr-un URI (*Universal Resource Identifier*).

Interfaţa standard de programare a unui serviciu REST este *Java API for XML Restful Services - JAX-WS.*

Transferul resursei către un client şi prelucrarea resursei se face prin operaţii indicate de antetele mesajului `http` *GET, POST, PUT, DELETE, etc.*

Resursele sunt fără stare iar modelul de aplicaţie este cel de client-server.

Pentru sistemelele care satisfac aceste restricţii se utilizează terminologia *RESTful.*

Principalul exemplu de sistem RESTful este World Wide Web (WWW) cu protocolul Hyper Text Transfer Protocol (HTTP).

Serviciile Web bazate pe REST se bazează pe accesul la resurse - definite prin identificatori şi nu pe apelarea unor metode, ca în modelul Remote Procedure Call (RPC).

Standardul *JAX-RS Java API for RESTful Web Services* a ajuns la versiunea 2. (Versiunea 1 este definită de JSR 311, iar versiunea 2 este definită de JSR 339)

În prezent, în Java, există mai multe implementări pentru servicii REST.

10.1 Descrierea unui serviciu

Spre deosebire de aplicaţiile bazate pe apelul de procedură la distanţă (RMI, CORBA,) unde prezentarea ofertei se face printr-o interfaţa Java, într-un serviciu JAX-WS, JAX-RS descrierea sau specificarea acestuia se realizează prin sublimbaje `xml`:

- *Web Service Description Language - WSDL* pentru servicii JAX-WS.

  Descrierea datelor din mesajele vehiculate se face prin *XML Schema,* de asemenea un sublimbaj `xml.`

- *Web Application Description Language - WADL* pentru servicii JAX-RS.

Mesajele dintre client şi server folosesc protocolul de reprezentare *Simple Object Access Protocol - SOAP* care este independent de platforma de calcul şi de limbajul de programare. SOAP este tot un sublimbaj `xml,` standard *World Wide Web Consortium - W3C.* În instrumentele actuale de dezvoltare, SOAP este transparent programatorului, dar Java ofera suport de programare prin pachetul `javax.xml.soap.`

Produsul *soapUI* oferă posibilitatea vizualizării mesajelor SOAP ale serviciilor Web pornind de la descrierea acestora prin WSDL, respectiv WADL.

10.2 Modelul JAX-WS prin *Metro*

JSR (Java Specification Request) 109 defineşte o interfaţă de programare (API) pentru realizarea serviciilor Web bazate pe RPC : *Java API for XML Web Services (JAX-WS).* Un asemenea serviciu Web se poate implementa prin:

- servlet:

  Serviciul este implementat ca o clasă Java care rulează într-un container Web, fiind integrat într-un servlet. Integrarea este complet transparentă programatorului.

- sesiune EJB (Enterprise Java Bean) fără stare (stateless session):

  Serviciul rulează într-un container EJB.

10.2.1 Serviciu JAX-WS ca servlet

Cadrul de lucru pe care îl vom utiliza este *Metro,* dezvoltat de *Oracle.* Un scop al cadrului de lucru *Metro* este asigurarea interoperabilităţii între server şi client atunci când acestea sunt realizate pe platformele soft Java şi .NET, dar facilitează şi dezvoltarea în mediul omogen Java. *Metro* implementează modelul JAX-WS.

Alternativ, s-ar fi putut folosi implementarea de referinţă *jaxws-ri,* dezvoltat tot de *Oracle* sau *apache-CXF.*

Metro oferă suport pentru dezvoltarea serviciului Web pe serverele:

- *apache-tomcat;*[2]
- *glassfish.*

Instalarea în *apache-tomcat* se face

- prin `ant` cu fişierul *metro-on-tomcat.xml* aflat în distribuţia lui *Metro.*
- fişierele `jar` aflate în catalogul *metro\lib* se copiază în *apache-tomcat-*\lib* sau în catalogul `WEB-INF\lib` al serviciului.

Dezvoltarea aplicaţiei server

Clasa serverului este o clasă POJO cu adnotări specifice. Definirea serviciului, a operaţiilor pe care le oferă serviciul şi a parametrilor de intrare pentru fiecare operaţie se face utilizând *adnotările* `@WebService,` respectiv `@WebMethod` şi `@WebParam.`

Vom dezvolta serviciul Web pentru calculul unei integrale. În clasa server se va crea câte o instanţă a claselor *mathlib.client.cvadra.JepDataIn* şi *mathlib.client.cvadra.impl.MetodaSimpsonWeb* a mini-bibliotecii *mathlib* iar rezultatele se obţin apelând metoda de integrare numerică *metodaSimson.* În final, sursa devine:

```
1  package integrala.server;

3  import javax.jws.WebMethod;
4  import javax.jws.WebParam;
5  import javax.jws.WebService;
6  import mathlib.client.cvadra.*;
7  import mathlib.client.cvadra.impl.MetodaSimpsonWeb;

9  @WebService()
10 public class MetodaSimpsonWS {
```

[2]Funcţionează şi în serverul Web `jetty`.

```
12   @WebMethod(operationName = "integreaza")
13   public DataOut integreaza(@WebParam(name = "a") String a,
14     @WebParam(name = "b") String b,
15     @WebParam(name = "svar") String svar,
16     @WebParam(name = "expr") String expr,
17     @WebParam(name = "eps")  String eps,
18     @WebParam(name = "nmi")  String nmi) {

20     JepDataIn din=new JepDataIn(svar,expr);
21     din.setA(a);
22     din.setB(b);
23     din.setEps(Double.parseDouble(eps));
24     din.setNmi(Integer.parseInt(nmi));
25     IMetodaSimpson obj=new MetodaSimpsonWeb();
26     DataOut dout=obj.metodaSimpson(din);
27     return dout;
28   }
29 }
```

Vom considera structura

```
myapp
|--> src
|      |--> mypackage
|      |      |--> server
|      |      |      |  *.java
|--> war
|      |--> WEB-INF
|      |      |--> classes
|      |      |--> lib
|      |      |      web.xml
|      |      |      sun-jaxws.xml
```

Compilarea se face prin intermediul utilitarului `apt - Annotation Proccessing Tool` din *jdk* prin intermediul unei sarcini date de clasa `com.sun.tools.ws.ant.Apt` din Metro. Codul corespunzător din `build.xml` este

```
<taskdef name="apt" classname="com.sun.tools.ws.ant.Apt">
      <classpath refid="myclasspath"/>
</taskdef>

<target name="build-server" depends="init">
      <apt
        fork="true"
        debug="true"
        destdir="war/WEB-INF/classes"
        sourcedestdir="war/WEB-INF/classes"
        sourcepath="src">
        <classpath>
          <path refid="myclasspath"/>
        </classpath>
        <source dir="src">
          <include name="**/server/*.java"/>
        </source>
      </apt>
 </target>
```

Pe baza claselor din `source`=*src\mypackage\server* se vor genera în `sourcedestdir` o serie de clase iar rezultatul compilării se depun în catalogul indicat de `destdir.`

În cazul exemplului considerat aceste clase sunt *integrala.server.jaxws.Integreaza.java* şi *integrala.server.jaxws.IntegreazaResponse.java.*

Sunt necesare fişierele:

- *sun-jaxws.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints xmlns='http://java.sun.com/xml/ns/jax-ws/ri/runtime' version='2.0'>
    <endpoint
        name='jaxws-integrala'
        implementation='integrala.server.MetodaSimpsonWS'
        url-pattern='/integrala'/>
</endpoints>
```

- *web.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
  <listener>
    <listener-class>
      com.sun.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>integrala</servlet-name>
    <servlet-class>com.sun.xml.ws.transport.http.servlet.WSServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>integrala</servlet-name>
    <url-pattern>/integrala</url-pattern>
  </servlet-mapping>
  <session-config>
    <session-timeout>60</session-timeout>
  </session-config>
</web-app>
```

Se reţin următoarele corelaţii ale denumirilor:

- *url-pattern* fixat în *web.xml* este redeclarat în fişierele *sun-jaxws.xml.*
- Numele serviciului declarat în *sun-jaxws.xml* trebuie să fie numele arhivei `war`.

  Pentru exemplul nostru, *wsdl*-ul serviciului va fi disponibil la

  http://localhost:8080/jaxws-integrala/integrala?wsdl

Dezvoltarea aplicaţiei client. Realizarea aplicaţiei client presupune ca serviciul să fie activ pe serverul Web.

Dezvoltarea părţii (aplicaţiei) client începe cu generarea unor clase care mijlocesc apelarea serviciului. Generarea se face utilizând utilitarul `wsimport` din `jdk` pe baza accesării fişierului `wsdl` asociat serviciului.

Codul obiectivului din `build.xml` este

```
<target name="generate-client" depends="init">
    <exec executable="${env.JAVA_HOME}/bin/wsimport">
        <arg value="-d"/>
        <arg value="${build.dir}"/>
```

```
        <arg value="-keep"/>
        <arg value="-p"/>
        <arg value="${app.name}.client"/>
        <arg value="${wsdl.uri}"/>
    </exec>
</target>
```

Opţiunea `-d` specifică locaţia unde se depun fişierele generate, opţiunea `-p` indică pachetul din care fac parte clasele generate.

Dintre aceste clase, în codul clientului propriu-zis se foloseşte clasa *MetodaSimpsonWSService.* Numele clasei s-a obţinut adăugând sufixul *Service* la numele clasei server. Această clasă conţine metoda *getMetodaSimpsonWSPort()* ce returnează un reprezentant al serviciului pe calculatorul clientului.

Astfel referinţa la serviciu se obţine prin

```
MetodaSimpsonWSService service=new MetodaSimpsonWSService();
MetodaSimpsonWS port=service.getMetodaSimpsonWSPort();
```

Prin variabila *port* putem apela orice operaţie a serviciului.

Codul clientului este

```
1  package integrala.client;
2  import java.util.Scanner;

4  public class ClientIntegrala {
5    public static void main(String[] args) {
6      MetodaSimpsonWSService service=new MetodaSimpsonWSService();
7      MetodaSimpsonWS port=service.getMetodaSimpsonWSPort();
8      Scanner scanner=new Scanner(System.in);
9      System.out.println("Extremitatea stanga");
10     String a=scanner.next();
11     System.out.println("Extremitatea dreapta");
12     String b=scanner.next();
13     System.out.println("Simbolul variabilei");
14     String svar=scanner.next();
15     System.out.println("Expresia de integrat");
16     String expr=scanner.next();
17     System.out.println("Toleranta");
18     String eps=scanner.next();
19     System.out.println("Numar maxim admis de iteratii");
20     String nmi=scanner.next();
21     DataOut dout=port.integreaza(a,b,svar,expr,eps,nmi);
22     System.out.println("Indicatorul de raspuns : "+dout.getInd());
23     System.out.println("Integrala : "+dout.getIntegrala());
24     System.out.println("Numar de iteratii efectuate : "+dout.getNi());
25   }
26 }
```

Instrucţiunea *port.integreaza(...)* este blocantă, fapt pentru care această variantă de program client este denumită sincronă. Interfaţa de programare JAX-WS permite construirea de clienţi asincroni, caz în care apelarea unei operaţii returnează obiecte de tip `java.util.concurrent.Future` şi este neblocantă.

10.3 Modelul JAX-RS prin *jersey*

Entităţile utilizate sunt:

- *Clasă resursă - Resource class.* Resursa Web este reprezintă de o clasă Java cu adnotări JAX-RS. *Clasa resursă rădăcină - Root resource class.* Clasă cu adnotarea `@Path`. Resursele adiacente se definesc relativ la această clasă (resursă).
- *Metoda de identificare a cererii - Request method designator.* Adnotarea `@GET / @POST /@PUT / @DELETE` este folosită pentru identificarea cererii HTTP în vederea desemnării metodei de generare / prelucrare a resursei.
- *Metodă de generare / prelucrare a resusei - Resource method.*
- *Localizator a resurselor adiacente - Sub-resource locator.* Metodă pentru localizarea a resurselor adiacente, adică a resurselor care se specifică relativ la resursa rădăcină.
- *Metoda de generare / prelucrare a unei resurse adiacente - Sub-resource method.*
- *Provider* o implementare a interfeţei JAX-RS.

O implementare de referinţă (*Reference Implementation* - RI) este oferită de pachetul *jersey-*.*.** realizat de *Oracle.*

Serverul Web care se va utiliza va fi *apache-tomcat.* În vederea desfăşurării, serviciul se arhivează, având extensia `war`. Numele arhivei va desemna numele serviciului. Structura care se arhivează va fi

```
catalogul_serviciului_RESTful
|--> WEB-INF
|    |--> classes
|    |    |--> resources
|    |    |    |   *.class
|    |--> lib
|    |    |    *.jar
|    index.html sau index.jsp
```

Printre fişierele **.class* se găsesc resursele serviciului. Fişierele **.jar* sunt cele din catalogul `lib` al distribuţiei *jersey* şi eventual cele cerute de aplicaţie. *index.html* sau *index.jsp* oferă oportunitatea apelării serviciului şi de obicei reprezintă un client Web.

Serviciul *RESTful* este oferit prin intermediul unui servlet

`org.glassfish.jersey.servlet.ServletContainer,`

complet transparent programatorului, specificat doar în fişierul *web.xml*:

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <!-- This web.xml file is not required when using Servlet 3.0 container,
3       see implementation details
4       http://jersey.java.net/nonav/documentation/latest/jax-rs.html -->
5  <web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
6    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
8    http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
9      <servlet>
10         <servlet-name>Jersey Web Application</servlet-name>
11         <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
```

```
12          <init-param>
13              <param-name>jersey.config.server.provider.packages</param-name>
14              <param-value>resources</param-value>
15          </init-param>
16          <load-on-startup>1</load-on-startup>
17      </servlet>
18      <servlet-mapping>
19          <servlet-name>Jersey Web Application</servlet-name>
20          <url-pattern>/resources/*</url-pattern>
21      </servlet-mapping>
22  </web-app>
```

Toate resursele trebuie să facă parte din în catalogul *resources.*

Problema de programare constau în:

- declararea unei resurse
  - adnotarea `@Path(`*"uriResursă"*`)` defineşte o resursă;
  - adnotările `@GET, @POST,` etc indică metoda care solicită resursa;
  - adnotarea `@Produces(String` *MIME-type*`)` indică tipul resursei;
- preluarea parametrilor pentru care există mai multe variante de programare:
  - prin context, indicat de adnotarea `@Context;`
  - prin adnotarea `@QueryParam` pentru metada `get` sau adnotarea `@FormParam` în cazul utilizării metodei `post;`
  - prin adnotarea `@PathParam;`
  - prin componente java, datele sunt cuprinse într-o componentă Java (o clasă POJO - *Plain Old Java Object*) care este convertită în mesaj `XML` sau `JSON`. Există mai multe implementări ale acestei tehnologii care solicită resurse suplimentare distribuţiei *jersey* (*MoXy POJO, jackson*).
- generarea resursei se obţine prin intermediul claselor:
  - `javax.ws.rs.core.Response`

    Metoda `public static Response.ResponseBuilder ok(Object` *method,* `String` *MIME-type*`)` generează resursa prin intermediul metodei *method.*
  - `javax.ws.rs.core.Response.ResponseBuilder`

    Metoda `public abstract Response build()` va returna obiectul `Response` în urma executării metodei `ok.`

  Semnalăm posibilitatea programării asincrone a generării resursei.

Exemplificăm din nou cu calculul unei integrale utilizând resursele mini-bibliotecii *mathlib.* Serviciul va putea returna rezultatul sub forma de *text/plain* sau *text/html.*

Codul serviciului este:

```
1  package resources;
2  import javax.ws.rs.QueryParam;
3  import javax.ws.rs.core.Response;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.GET;
6  import mathlib.client.cvadra.JepDataIn;
7  import mathlib.client.cvadra.DataOut;
8  import mathlib.client.cvadra.IMetodaSimpson;
9  import mathlib.client.cvadra.impl.MetodaSimpsonWeb;

11 @Path("integ")
12 public class IntegResource {
13   private DataOut dout;
14   private double s=Double.NaN;

16   public IntegResource() {}

18   @GET
19   public Response doGet(
20     @QueryParam("svar") String svar,
21     @QueryParam("expr") String expr,
22     @QueryParam("a") String a,
23     @QueryParam("b") String b,
24     @QueryParam("nmi") String nmi,
25     @QueryParam("eps") String eps,
26     @QueryParam("tip") String tip){
27     JepDataIn din=new JepDataIn(svar,expr);
28     din.setA(a);
29     din.setB(b);
30     din.setEps(Double.parseDouble(eps));
31     din.setNmi(Integer.parseInt(nmi));
32     IMetodaSimpson obj=new MetodaSimpsonWeb();
33     dout=obj.metodaSimpson(din);

35     int ntip=0;
36     if(tip.equals("text/html")) ntip=1;
37     Response r=null;
38     switch(ntip){
39       case 0:
40         r=Response.ok(getPlainRep(),"text/plain").build();
41         break;
42       case 1:
43         r=Response.ok(getHtmlRep(),"text/html").build();
44         break;
45     }
46     return r;
47   }

49   public String getPlainRep() {
50     String rezultat="";
51     rezultat=rezultat+
52       "Indicatorul de raspuns : "+ (new Integer(dout.getInd())).toString()+"\n";
53     rezultat=rezultat+
54       "Integrala : "+ (new Double(dout.getIntegrala())).toString()+"\n";
55     rezultat=rezultat+
56        "Numar de iteratii efectuate : "+ (new Integer(dout.getNi())).toString();
57     return rezultat;
58   }

60   public String getHtmlRep() {
61     String rezultat="<html><head></head><body bgcolor=
62      \"#bbeebb\"><center><h1>Rezultatul furnizat de serviciul RESTfull";
63     rezultat=rezultat+"<table border=\"1\">";
64     rezultat=rezultat+"<tr>";
65     rezultat=rezultat+"<td> Indicatorul de raspuns </td>";
66     rezultat=rezultat+"<td>"+(new Integer(dout.getInd())).toString()+"</td>";
```

```
67      rezultat=rezultat+"</tr>";
68      rezultat=rezultat+"<tr>";
69      rezultat=rezultat+"<td> Integrala </td>";
70      rezultat=rezultat+"<td>"+(new Double(dout.getIntegrala())).toString()+"</td>";
71      rezultat=rezultat+"</tr>";
72      rezultat=rezultat+"<tr>";
73      rezultat=rezultat+"<td> Numarul iteratiilor efectuate </td>";
74      rezultat=rezultat+"<td>"+(new Integer(dout.getNi())).toString()+"</td>";
75      rezultat=rezultat+"</tr>";
76      rezultat=rezultat+"</table></h1></center></body></html>";
77      return rezultat;
78    }
79  }
```

Apelarea serviciului *RESTful* dintr-un navigator (clientul Web) se obţine cu

```
1  <html>
2    <body bgcolor="#bbeebb" >
3      <center>
4        <h1> Calculul unei integrale printr-un serviciu RESTful </h1>
5        <form method="get"
6              action="/integrala/resources/integ">
7         <p>Introduce&#355;i</p>
8         <table border="1">
9         <tr>
10          <td> Simbolul variabilei </td>
11          <td> <input type="text" name="svar" size="10" />
12        </tr>
13        <tr>
14          <td> Expresia func&#355;iei de integrat </td>
15          <td> <input type="text" name="expr" size="10" />
16        </tr>
17        <tr>
18          <td> Extremitatea st&#226;g&#259; </td>
19          <td> <input type="text" name="a" size="10" />
20        </tr>
21        <tr>
22          <td> Extremitatea dreapt&#259; </td>
23          <td> <input type="text" name="b" size="10" />
24        </tr>
25        <tr>
26          <td> Num&#259;rul maxim admis de itera&#355;ii </td>
27          <td> <input type="text" name="nmi" size="10" value="50" />
28        </tr>
29        <tr>
30          <td> Toleran&#355;a </td>
31          <td> <input type="text" name="eps" size="10" value="1e-6" />
32        </tr>
33        <tr>
34          <td> Selecta&#355;i tipul r&#259;spunsului </td>
35          <td>
36             <select name="tip">
37                <option value="text/plain"> Text/Plain </option>
38                <option value="text/html"> Text/Html </option>
39             </select>
40          </td>
41        </tr>
42        <tr align="center">
43          <td> <input type="submit" value="Calculeaza" /> </td>
44        </tr>
45       </table>
46       </form>
47     </center>
48     </body>
49  </html>
```

Produsul *jersey* oferă o modalitate simplă de programare a unui client pentru un serviciu RESTful:

```
1  package integ;
2  import javax.ws.rs.client.Client;
3  import javax.ws.rs.client.ClientBuilder;
4  import javax.ws.rs.client.WebTarget;
5  import java.util.Scanner;

7  public class JerseyClient {
8    public static void main(String args[]) {
9      Client client = ClientBuilder.newClient();
10     String rootURL="http://localhost:8080/integrala/resources/integ";
11     WebTarget webTarget = client.target(rootURL);
12     Scanner scanner=new Scanner(System.in);
13     System.out.println("Extremitatea stanga");
14     String a=scanner.next();
15     System.out.println("Extremitatea dreapta");
16     String b=scanner.next();
17     System.out.println("Simbolul variabilei");
18     String svar=scanner.next();
19     System.out.println("Expresia de integrat");
20     String expr=scanner.next();
21     System.out.println("Toleranta");
22     String eps=scanner.next();
23     System.out.println("Numar maxim admis de iteratii");
24     String nmi=scanner.next();
25     String response=webTarget.
26       queryParam("svar",svar).
27       queryParam("expr",expr).
28       queryParam("a",a).
29       queryParam("b",b).
30       queryParam("nmi",nmi).
31       queryParam("eps",eps).
32       queryParam("tip","text/plain").
33       request().get(String.class);
34     System.out.println(response);
35   }
36 }
```

Partea II

Programare paralelă în Java

Capitolul 11

Introducere în programarea concurentă / paralelă

11.1 Procese paralele

Evenimentele se petrec în spaţiu şi timp. Există următoarele posibilităţi: două evenimente se petrec în acelaşi loc la alte momente de timp (succesiv, secvenţial) sau cele două evenimente se petrec în acelaşi timp, dar în locuri diferite (concomitent, concurent, paralel). Astfel paralelismul este o trăsătură a acestui univers.

Apare astfel naturală cerinţa ca un calculator să poată simula sau exe- cuta operaţii, activităţi paralele.

În cele ce urmează printr-un *proces* se va înţelege un şir de acţiuni executate câte una singură la un moment dat, iar un *procesor* va indica un dispozitiv care execută instrucţiuni în mod secvenţial.

Mai precis un proces este caracterizat de următoarele atribute (BÂSCĂ O.C., POPESCU I., 1989):

- *indivizibilitatea:* Procesul este o unitate atomică care nu mai este divizat în alte unităţi cărora sistemul de calcul să le aloce resurse în mod autonom.
- *secvenţialitatea:* Operaţiile unui proces se execută câte una singură la un moment dat.
- *asincronismul:* Un proces se desfăşoară independent de celelalte procese, cu excepţia momentelor de interacţiune cu alte procese.
- *temporalitatea:* Un proces există, se manifestă şi este recunoscut numai în intervalul de timp dintre lansarea sa în execuţie şi terminarea sa.

Într-un program, un proces are o existenţă formată din mai multe stări:

- Trecerea de la starea *inexistent* la starea *creat* se face prin declararea procesului.
- Din starea *creat* un proces ajunge în starea *executabil* prin lansarea în execuţie sau printr-o formă de activare. Starea *executabil* are două substări:

1. starea *gata de executare*: procesul este pregătit pentru executare, dar nu i s-a alocat încă un procesor logic/fizic;
2. starea *în curs de executare*: procesului i s-a alocat un procesor logic/fizic care îi execută instrucţiunile.

- La terminarea executării unui proces, acesta trece în starea *terminat.*

  În final procesele sunt distruse - automat sau prin instrucţiuni - trecându-se în starea *inexistent.*

Un proces se poate *bloca,* mai precis poate fi

- *amânat* : întreruperea pentru un anumit timp a execuţiei procesului.
- *suspendat* : întreruperea execuţiei pe baza unei relaţii cu alte procese. Procesul poate reveni ulterior în starea de *executabil,* reluând executarea instrucţiunilor din momentul suspendării. Trecerea în starea *executabil* se realizează fie aleator, fie cu disciplina unei cozi.

Un calculator cu procesare paralelă poate fi definit, într-o primă aproximaţie, ca o maşină capabilă să execute diferite operaţii utilizând simultan cel puţin două procesoare.

Merită de semnalat aici **clasificarea lui Flynn:** Din punctul de vedere al secvenţei de instrucţiuni executate de către un sistem de calcul şi al secvenţei de date asupra căreia acţionează instrucţiunile la un moment dat, se pun în evidenţă următoarele modele de calculatoare:

- SISD : Single Instruction, Single Data stream;
- SIMD : Single Instruction, Multiple Data stream;
- MIMD : Multiple Instruction, Multiple Data stream.

Modelul SISD corespunde sistemelor de calcul obişnuite, iar modelele SIMD şi MIMD caracterizează sistemele de calcul ce oferă facilităţi de calcul paralel. În modelul SIMD toate procesoarele efectuează aceleaşi operaţii dar pe mulţimi de date diferite, în timp ce în modelul MIMD procesoarele efectuează secvenţe distincte de operaţii asupra unor mulţimi de date.

Dacă execuţia proceselor se efectuează concomitent pe procesoare diferite atunci execuţia se numeşte *paralelă.*

Dacă execuţia proceselor se efectuează cu un singur procesor atunci execuţia se numeşte *concurentă.*

Preţul de achiziţie, întreţinere şi exploatare al unui calculator paralel fiind mare, pentru orice instituţie problema achiziţionării unui asemenea calculator se pune în termenii raportului eficienţă/cost. Necesitatea de a dispune de o putere de calcul cât mai mare la preţuri cât mai mici a condus la soluţii care permit programare paralelă într-o reţea obişnuită de calculatoare.

În cazul în care procesele se execută pe calculatoare distincte, legate într-o reţea, atunci execuţia se numeşte *distribuită, paralel - distribuită.*

Un grup de calculatoare interconectate având un singur punct de administrare şi ale căror resurse sunt unite pentru rezolvarea unei probleme formează un *cluster*.

Un grup de calculatoare interconectate, fiecare cu posibilitatea de administrare proprie care îşi reunesc resursele pentru la rezolvarea unei probleme formează un *grid*.

De multe ori diferenţa dintre cluster şi grid este mai mult de natura politicii de utilizare decât de natură tehnică.

Dacă calculatoarele aparţin unui intranet se mai utilizează terminologia de *reţea de staţii de lucru* (*network of workstations*), iar dacă calculatoarele sunt conectate prin internet atunci se utilizează termenul de *grid*.

O memorie partajabilă se va considera că este de tip *PRAM* (Parallel Random Access Memory). Modelul PRAM presupune că toate procesele au acces la respectiva memorie comună, fiecare proces comunică în ambele sensuri cu memoria partajabilă în sensul că poate scrie şi citi date în locaţiile memoriei.

Reglementarea accesului la memoria comună se face conform disciplinei *CREW* (Concurrent Read; Exclusive Write). Potrivit acestei restricţii, este permisă citirea în paralel (concomitent) din aceaşi locaţie de memorie de către două sau mai multe procese, dar scrierea concomitentă în aceaşi locaţie este interzisă.

Programarea paralelă şi paralel-distribuită apare ca o combinaţie a două tipuri de activităţi:

- de calcul, specific aplicaţiei, executată de procese;
- de coordonare, responsabilă de efectuarea comunicaţiilor şi de lansare a proceselor.

Există mai multe încercări de definire a *coordonării* (PAPADOPOULOS G.A., ARBAB F., 1998):

- coordonarea constă în gestionarea dependinţelor dintre activităţi;
- coordonarea este procesul de construire a programelor prin lipirea proceselor.

Un *model de coordonare* este liantul prin care sunt lipite activităţile separate (procesele) într-un ansamblu.

Din punctul de vedere al programării, modelele de coordonare aparţin uneia din categoriile:

- Coordonare condusă prin date (*data-driven*) – evoluţia în orice moment se decide în funcţie de datele de expediat sau de cele recepţionate. Coordonatorul este responsabil de examinarea şi manipularea datelor necesare coordonării proceselor.

  În acestă categorie intră aplicaţiile bazate pe o memorie comună, partajată.

- Coordonarea condusă prin sarcini sau orientată pe procese (*control driven / task driven / process oriented*). În acest caz lansarea proceselor este independentă de date. Aplicaţiile bazate pe *Communicating Sequential Processes* (CSP) - dezvoltat de C.A.R. Hoare aparţin acestei categorii: JCSP.

11.2 Probleme specifice calculului paralel

Pentru a înţelege problemele care se ridică la realizarea unui program în care există procese ce se execută în paralel considerăm două exemple deosebit de importante.

1. Se consideră un sistem de rezervare de bilete în care de la două terminale se poate cere ocuparea unor locuri. Acţiunile necesare rezervării sunt executate de procesele P_1 şi P_2 care se desfăşoară în paralel; permiţând solicitarea de bilete în orice moment, de la oricare terminal. Fiecare operaţiune de rezervare impune, printre altele, şi incrementarea unei variabile v, care indică numărul total de locuri ocupate.

2. Fie un sistem format din două procese P_1 şi P_2, primul producând nişte date care apoi vor fi preluate de cel de al doilea. Comunicarea între cele două procese se realizează prin intermediul unei zone de memorie – tampon, – de o anumită dimensiune, în care procesul P_1 introduce informaţii (sub forma unor înregistrări), pe care apoi P_2 le extrage.

   Procesul P_2 nu este nevoit să aştepte pentru fiecare înregistrare în parte, iar P_1 nu aşteptă până ce P_2 este gata să recepţioneze înregistrarea produsă. Procesele evoluează independent unul de celă- lalt, P_1 introducând în tampon înregistrarea produsă, de unde P_2 o va extrage la nevoie.

   Se impun următoarele restricţii:

   - Procesul P_1 încearcă să introducă o înregistrare în tampon şi constată că acesta s-a umplut. În acest caz, el va trebui să aştepte până ce se iveşte un loc în tampon (ceea ce se va întâmpla ca urmare a extrageri unei înregistrări de către P_2).
   - Procesul P_2 încearcă să extragă o înregistrare şi constată că tamponul este gol. În acest caz el va trebui să aştepte până ce apare o înregistrare ca urmare a introduceri unei înregistrări în tampon de către P_1.

Din aceste exemple se deduce că între procese există interacţiuni de forma:

- **comunicări:** transmitere de informaţii între procese;
- **sincronizări:** restricţii asupra evoluţiei în timp a unui proces.

Problemele de sincronizare sunt:

1. **Excluderea reciprocă (mutuală) :** Forma de sincronizare prin care se evită utilizarea simultană de către mai multe procese a unei *resurse critice.* O resursă este critică dacă, la un moment dat, poate fi utilizată doar într-un proces.

   Variabila v din Exemplul 1 este o resursă critică. Procesele P_1 şi P_2 nu pot accesa simultan această variabila, altfel spus, cele două procese trebuie să se sincronizeze prin excludere reciprocă.

2. **Sincronizarea pe condiţie :** Forma de sincronizare prin care se amână executarea sau continuarea executării unui proces până când o anumită condiţie devine adevărată.

   Procesele P_1 şi P_2 ale Exemplului 2 trebuie să se sincronizeze în cazul în care tamponul este plin, respectiv gol.

Un caz particular al sincronizării pe condiţie este sincronizarea **barieră**. Mai multe procese execută aceaşi prelucrare, între care există un marcaj - care reprezintă bariera. Un proces ce ajunge la marcaj se blochează până în momentul în care toate procesele ating marcajul, după care se deblochează şi îşi continuă execuţia.

Sistemul de programare paralelă / concurentă reprezintă o interfaţă (software şi/sau hardware) între program şi sistemul de calcul, el furnizând printre altele, *primitive* pentru comunicare între procese şi pentru sincronizarea lor. Primitivele de sincronizare sunt operaţiile pe care sistemul de programare paralelă / concurentă le pune la dispoziţia programatorului în vederea rezolvării problemelor de sincronizare. Practic, aceste primitive se prezintă programatorului sub forma unor instrucţiuni oarecare, pe care programatorul le poate folosi fără a cunoaşte modul lor de implementare.

Rezultatele unui program executat de un sistem de calcul în care există procese paralele trebuie să fie independent de vitezele relative de execuţie ale proceselor şi de ordinea în care acestea sunt executate, atunci când nu sunt supuse unor restricţii de precedenţă.

Modelul de aplicaţie utilizat în cele mai multe cazuri este cel de *dispecer - lucrător* (master - slave). O componentă a aplicaţiei - dispecerul - coordonează şi distribuie o serie de activităţi de calcul lucrătorilor. După efectuarea calculelor, lucrătorii transmit rezultatele dispecerului, care le utilizează pentru finalizarea rezolvării problemei.

Acest model de aplicaţie este opusul modelului *client - server,* în care clienţii solicită serverului efectuarea unor prelucrări. De data asta, aplicaţia server este cel care efectuează operaţiile de prelucrare.

S-au pus în evidenţa o serie de modele (tipuri) de programe / aplicaţii specifice calculului paralel:

- *Program SPMD* (Single Program, Multiple Data) – un program acţionează simultan, prin acelaşi cod, asupra datelor în procese distincte.

- *Modelul nodal* – o instanţă a aceluiaşi program este executat de fiecare proces.

În vederea executării unei aplicaţii paralel - distribuită într-o reţea de staţii de lucru trebuie efectuate operaţiile:

- definirea reţelei de staţii de lucru prin instalarea şi lansarea în execuţie a suportului soft utilizat;

- *desfăşurarea aplicaţiei* (deployment), adică instalarea aplicaţiei împre- ună cu resurselor necesare pe fiecare calculator al reţelei;

- lansarea în execuţie a aplicaţiei - a lucrătorilor şi a dispecerului.

11.3 Eficienţa programelor paralele

Atribute de calitate ale programelor. În multe domenii concurenţa devine cerinţa majoră a algoritmilor şi programelor. De aici şi importanţa *scalabilităţii* unui program, adică proprietatea/atributul său de a se adapta eficient la creşterea numărului de procesoare. Un program care foloseşte numai un număr fixat de procesoare **nu** este un program performant, la fel ca un program care poate rula doar pe un tip de calculatoare – aspect ce vizează *portabilitatea.*

Scalabilitatea şi portabilitatea sunt atribute importante ale unui produs informatic.

Într-un sistem multiprocesor sau distribuit, în cadrul unui proces costul/timpul de acces la memoria propriului procesor - read/write - este mai mic decât la memoria altui procesor, realizat prin transmisii de mesaje - send/receive -. În acest fel punem în evidenţă proprietatea *localizării,* de a păstra un raport convenabil între cele două feluri de acces la resurse.

Alături de concurenţă, scalabilitate, localizare, o altă trăsătură a programelor paralele este *modularitatea.* Pentru stăpânirea şi gestionarea fenomenelor legate de execuţia, comunicaţii, sincronizări, etc. programele se alcătuiesc din procese (module).

Dacă rezolvarea unei probleme utilizează p procese, iar execuţia procesului i are loc în t_i unităţi de timp, $i \in \{1, 2, \ldots, p\}$ atunci vorbim de *încărcare echilibrată a procesoarelor* dacă

$$\beta = \frac{t_{mediu}}{\max_{1 \le i \le p} t_i} \approx 1,$$

unde $t_{mediu} = \frac{1}{p} \sum_{i=1}^{p} t_i$. Numărul β se numeşte *indice de încărcare a proceselor.*

Indice de performanţă. Evaluarea performanţelor unui program se poate face prin *indicele de performanţă,* definit ca numărul de operaţii în virgulă mobilă efectuate în unitatea de timp. Indicele de performanţă se exprimă în *Flops:* 1 Flops=1 operaţie / 1 s; 10^6 Flops= 1 MFlops; 10^9 Flops=1 GFlops= 1 Cray.

Exemplul 11.3.1

Fie şirul $(a_i)_{i \in \mathbb{N}}$ şi $\mathcal{P}$ un program care calculează suma

$$S_n = \sum_{i=1}^{n} a_i$$

în t_n secunde.

Pentru calculul lui S_n sunt necesare $n-1$ adunări, astfel indicele de performanţă este $\frac{n-1}{t_n}$.

În analiza indicelui de performanţă, prezintă interes determinarea valorii lui n_c de la care $t_n \ge 1$ şi în plus t_n variază "proporţional" cu n.

În Java măsurarea timpului face apel la metoda clasei `System`

```
static long currentTimeMillis()
```

care returnează numărul de milisecunde de la data de 1 ianuarie 1970.

Subliniem *principiul de incertitudine a calculatorului:* nu se poate măsura timpul de calcul al unui program cu precizie oricât de mică folosind chiar calculatorul pentru efectuarea măsurătorii.

Un alt indicator util este raportul *lucru / memorie* - ρ_{WM}. Se numeşte *lucru* numărul de operaţii în virgulă mobilă efectuate de un program, iar cantitatea de *memorie* este dat de numărul de locaţii de memorie necesare programului pentru datele de intrare, pentru reţinerea rezultatelor parţiale şi a celor finale.

În cazul Exemplului 11.3.1 sunt necesare $n+1$ locaţii de memorie, deci $\rho_{WM} = \frac{n-1}{n+1}$.

Exemplul 11.3.2

Fie $A \in M_n(\mathbb{R}),\ x, y \in \mathbb{R}^n, y = Ax$. Dacă $y_i = \sum_{i=1}^{n} a_{i,j}x_j$ atunci pentru calculul componentei y_i sunt necesare n înmulţiri şi $n-1$ adunări. Rezultă $\rho_{WM} = \frac{n(2n-1)}{n^2+2n}$.

Teorema 11.3.1 *Dacă între unitatea centrală şi memorie se pot transfera în unitatea de timp μ numere în virgulă mobilă, atunci indicele de performanţă este majorat de $\mu\rho_{WM}$.*

Notând cu w lucrul şi cu m cantitatea de memorie utilizată de un program, durata de execuţie t este mai mare decât durata transferului $t_{transfer}$ a celor m date între unitatea centrală şi memorie.

$$t \geq t_{transfer} \geq \frac{m}{\mu}.$$

Pentru indicele de performanţă rezultă

$$\frac{w}{t} \leq \mu\frac{w}{m} = \mu\rho_{WM}.$$

Eficienţa programelor paralele. Corespunzător rezolvării unei probleme introducem variabilele:

- T_0 timpul celui mai performant program/algoritm utilizat pe un sistem uniprocesor;
- T_p timpul folosit de programul/algoritmul paralel utilizând un sistem de calcul cu p procesoare.

Eficienţa unui program/algoritm se poate evalua cu indicatorii

1. $S_p = \frac{T_0}{T_p}$ *acceleraţia paralelă.* Are loc inegalitatea $S_p \leq p$. Într-adevăr, dacă presupunem prin absurd că $S_p = \frac{T_0}{T_p} > p \Leftrightarrow pT_p < T_0$ atunci utilizând acel algoritm şi folosind doar un singur procesor am avea $T_1 < T_0$, ceea ce contrazice definiţia lui T_0.
2. $\tilde{S}_p = \frac{T_1}{T_p}$ *viteza algoritmică.* Acest indicator scoate în evidenţă efectele sincronizărilor şi ale comunicaţiilor. Ideal, valoarea acestui indicator ar fi p.
3. $E_p = \frac{\tilde{S}_p}{p}$ *eficienţa în cazul utilizării a p procesoare.*

S-au formulat diverse evaluări ale vitezei algoritmice care neagă (Amdahl G.M., 1967) şi respectiv validează (Gustafson J.L., 1988) oportunitatea construirii calculatoarelor paralele cu număr mare de procesoare.

Urmând prezentarea lui Shi Y. (*Reevaluating Amdahl's Law and Gustafson's Law*, `http://cgvr.cs.uni-bremen.de/teaching/mpar_literatur/`), fie

- t_S durata execuţiei părţii secvenţiale (executată de un singur procesor);
- $t_P(p)$ durata executării părţii paralele utilizând p procesoare.

Durata executării programului cu un sigur procesor este $T_1 = t_S + t_P(1)$, iar cu p procesoare, durata este $T_p = t_S + t_P(p)$.

Raportat la durata totală de execuţie a programului cu un singur procesor, fracţiunea părţii secvenţiale este

$$\beta_A = \frac{t_S}{t_S + t_P(1)},$$

şi raportat la durata totală de execuţie a programului cu p procesoare, fracţiunea părţii secvenţiale este

$$\beta_G = \frac{t_S}{t_S + t_P(p)}.$$

În funcţie de cele două exprimări, fracţiunea părţii paralele este

$$1 - \beta_A = \frac{t_P(1)}{t_S + t_P(1)} \quad \text{şi respectiv } 1 - \beta_G = \frac{t_P(p)}{t_S + t_P(p)}.$$

Apreciind că $t_P(p) = \frac{t_P(1)}{p}$, viteza algoritmică se exprimă prin

- *Legea lui Amdahl*

$$\tilde{S}_p = \frac{T_1}{T_p} = \frac{t_S + t_P(1)}{t_S + \frac{t_P(1)}{p}} = \frac{1}{\beta_A + \frac{1-\beta_A}{p}};$$

- *Legea lui Gustafson*

$$\tilde{S}_p = \frac{T_1}{T_p} = \frac{t_S + pt_P(p)}{t_S + t_P(p)} = \beta_G + p(1 - \beta_G).$$

Deoarece $\frac{1}{\beta_A + \frac{1-\beta_A}{p}} \leq \frac{1}{\beta_A}$ din legea lui Amdahl rezultă că $\tilde{S}_p \leq \frac{1}{\beta_A}$, adică viteza algoritmică este mărginită relativ la numărul de procesoare. Drept consecinţă, scopul paralelizării este rezolvarea unor probleme de dimensiune din ce în ce mai mari şi nu rezolvarea într-un timp cât mai scurt a unei probleme de dimensiune fixată.

Pe de altă parte, deoarece orice program paralel are o parte secvenţială, potrivit legii lui Gustafson, viteza algoritmică este funcţie crescătoare relativ la p şi $\tilde{S}_p < p$, adică programul paralel are viteza algoritmică subliniară, neputându-se obţine programe cu viteza algoritmică liniară ($\tilde{S}_p = p$) şi supraliniară ($\tilde{S}_p > p$).

Stabilirea acestor rezultate presupune faptul că atât în varianta secvenţială ($p = 1$) cât şi în varianta paralelă ($p > 1$), programul execută aceleaşi operaţii.

11.4 Programare paralelă în Java

Limbajul de programare Java permite execuţia *simultană* a mai multor activităţi prin intermediul *firelor de execuţie (thread).*

Spre deosebire de un proces, un fir de execuţie nu presupune execuţia sa de către un procesor dedicat şi toate firele de execuţie împart acelaşi spaţiu de adrese. Fiecare fir de execuţie are o stivă proprie pentru gestionarea metodelor apelate şi a variabilelor locale.

Având acelaşi spaţiu de adrese, firele de execuţie comunică prin variabile comune.

Utilizarea firelor de execuţie nu conduce la reducerea timpului de execuţie - mai ales dacă există o singură unitate centrală. Într-un program Java, rolul utilizării firelor de execuţie este separarea preocupărilor, adică simplificarea programării. Este mai uşor de scris o secvenţă de cod pentru fiecare activitate a algoritmului, executarea lor trecând în sarcina Maşinii Virtuale Java şi a sistemului de operare care simulează execuţia *simultană.*

Singura problemă care rămâne în sarcina programatorului este rezolvarea problemelor de sincronizare între firele de execuţie.

Chiar în cazul unui calculator cu un procesor având mai multe nuclee de calcul (*multicore*) gestionarea acestor nuclee nu cade în sarcina programatorului şi nu există nici o certitudine că execuţia este paralelă.

Astfel utilizarea firelor de execuţie conduce doar la execuţie concurentă.

Pe un calculator obişnuit execuţie paralelă se va obţine prin utilizarea procesorului grafic. Dacă metoda de calcul este iterativă atunci programarea va fi mai simplă prin utilizarea unui *algoritm paralel şi iterativ asincron.*

11.5 Metode numerice paralele

Scopul acestei secţiuni este prezentarea unor metode numerice care se pot paraleliza prin *descompunerea domeniului de calcul.*

Rezolvarea unei probleme constă din executarea a N sarcini de calcul identice. Mulţimea acestor sarcini, $\{0, 1, \ldots, N-1\}$, reprezintă domeniul de calcul. Pentru efectuarea calculelor se vor utiliza p procese.

Fiecare proces $i \in \{0, 1, \ldots, p-1\}$ va efectua cel mult $\lceil \frac{N}{p} \rceil$ sarcini, unde

$$\left\lceil \frac{N}{p} \right\rceil = \begin{cases} \frac{N}{p} & \text{dacă } \frac{N}{p} \in \mathbb{N}, \\ \left[\frac{N}{p}\right] + 1 & \text{dacă } \frac{N}{p} \notin \mathbb{N}, \end{cases}$$

iar $[a]$ reprezintă partea întreagă a lui a.

Distribuţia celor N sarcini la cele p procese se poate face:

- **ciclic:** sarcina j va fi executată de procesul $j \ mod \ p$, sau procesul i execută sarcinile $s \cdot p + i, \ s = 0, 1, \ldots$.

  Pseudocodul procesului i este:

```
Calc(i)
pentru s = 0 : ⌈N/p⌉ − 1  executa
|      j ← s · p + i
|      daca j ≤ N − 1  atunci
|      |     executa sarcina j
|      ◇
◇
stop
```

- **în bloc:** sarcina j va fi executată de procesul $j \; div \; \lceil \frac{N}{p} \rceil$, sau procesul i execută sarcinile $\lceil \frac{N}{p} \rceil \cdot i + s, \; s = 0, 1, \ldots, \lceil \frac{N}{p} \rceil - 1$.

  Pseudocodul procesului i este:

```
Calc(i)
pentru s = 0 : ⌈N/p⌉ − 1  executa
|      j ← ⌈N/p⌉ · i + s
|      daca j ≤ N − 1  atunci
|      |     executa sarcina j
|      ◇
◇
stop
```

Exemplul 11.5.1

Pentru $N = 10$ şi $p = 4$ distribuirea sarcinilor este

Cazul ciclic

sarcina	0	1	2	3	4	5	6	7	8	9
procesul	0	1	2	3	0	1	2	3	0	1

Cazul în blocuri

sarcina	0	1	2	3	4	5	6	7	8	9
procesul	0	0	0	1	1	1	2	2	2	3

Un algoritm căruia i se poate aplica paralelizarea prin descompunerea domeniului de calcul se numeşte *trivial paralel.*

Metodele numerice de care ne ocupăm constau în generarea unui şir de aproximaţii pe baza unei formule de recurenţă

$$x^{(k+1)} = F(x^{(k)}), \quad k \in \mathbb{N}$$

iar $x^{(k)} = (x_i^{(k)})_{1 \le i \le N}$. Paralelizarea se referă doar la calculul unei aproximaţii pe baza formulei de recurenţă: cele N componente se calculează în N procese.

Global un algoritm va genera un şir finit de aproximaţii potrivit unei reguli de oprire, iar schema de trecere la o nouă iteraţie se poate face sincron sau asincron. În modul sincron are loc o sincronizare barieră la sfârşitul calculului unei componente iar în modul asincron această sincronizare nu mai are loc.

Metoda Jacobi pentru sisteme algebrice de ecuaţii liniare

Fie $A \in M_n(\mathbb{R})$ astfel încât $a_{i,i} \neq 0, \forall i \in \{1, 2, \ldots, n\}$ şi $b \in \mathbb{R}^n$. Rezolvarea sistemului algebric de ecuaţii liniare $Ax = b$ prin metoda Jacobi constă în generarea şirului $(u^k)_{k\in\mathbb{N}}, u^k = (u_1^k, \ldots, u_n^k)^T$, prin formulele de recurenţă

$$u_i^{k+1} = \frac{1}{a_{i,i}} \left(b_i - \sum_{\substack{j=1 \\ j\neq i}}^{n} a_{i,j} u_j^k \right) \qquad i \in \{1, \ldots, n\}, \tag{11.1}$$

$k \in \mathbb{N}$, iar prima aproximaţie $u^0 = (u_1^0, \ldots, u_n^0)^T$ este un element din $\mathbb{R}^n$.

Dacă matricea A are diagonala dominantă, adică $\sum_{\substack{j=1 \\ j\neq i}}^{n} |a_{i,j}| < |a_{i,i}|,\ i = 1, 2, \ldots, n$, sau este simetrică şi pozitiv definită, atunci convergenţa procedeului (11.1) este asigurată.

Determinarea simultană a rădăcinilor unui polinom

Fie polinomul $P \in \mathbb{C}[X]$, $P(z) = z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n$. Notăm prin $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)^T$ vectorul format de rădăcinile polinomului P, pe care le considerăm distincte două câte două, deci simple.

O clasă de metode pentru determinarea simultană a rădăcinilor polinomului P este dată de formula de recurenţă

$$z^{(k+1)} = T(z^{(k)}), \qquad k \in \mathbb{N}$$

unde $T : \Omega \subseteq \mathbb{C} \leftarrow \mathbb{C}, T(z) = (T_1(z), \ldots, T_n(z))^T$ este un operator ce defineşte metoda.

Amintim metoda Durand-Kerner. Dacă $z = (z_1, \ldots, z_n)^T$ atunci operatorul T corespunzător metodei Durand-Kerner este

$$T_i(z) = z_i - \frac{P(z_i)}{\prod_{\substack{j=1,n \\ j\neq i}} (z_i - z_j)}, \qquad i \in \{1, \ldots, n\}.$$

Formula de recurenţă devene

$$z_i^{(k+1)} = z_i^{(k)} - \frac{P(z_i^{(k)})}{\prod_{\substack{j=1}}^{n}{}_{\substack{j\neq i}} (z_i^{(k)} - z_j^{(k)})}, \qquad i \in \{1, \ldots, n\}, \quad k \in \mathbb{N}.$$

Dacă aproximaţia iniţiala $z^{(0)}$ este aleasă într-o vecinătate convenabilă a lui α atunci şirul de aproximaţii $(z^{(k)})_{k\in\mathbb{N}}$ converge către α.

Capitolul 12

Algoritm paralel şi iterativ

Punem în evidenţă două clase de algoritmi paraleli şi iterativi. Prezentare se referă la calculul unui punct fix al unei funcţii.

Fie $T : D \subseteq \mathbb{R}^n \to \mathbb{R}^n$ o funcţie care admite un punct fix $x^* \in D,\ T(x^*) = x^*$.

Calculul punctului fix va utiliza metoda aproximaţiilor succesive

$$x^{k+1} = T(x^k), \tag{12.1}$$

pornind de la un element $x^0 \in D$.

Fie $r = \|x^0 - x^*\|$. Dacă

1. $\overline{B}(x^*, r) \subseteq D$;
2. $\|T(x) - T(x^*)\| \leq \|x - x^*\|, \quad x \in D,$

atunci $x_k \in D,\ \forall\ k \in \mathbb{N}$.

Varianta paralelă va utiliza $L \leq n$ procese (fire de execuţie). Procesul $i \in \{1, \ldots, L\}$ acţionează asupra variabilelor $(x_{j_1}, \ldots, x_{j_{n_i}}) = \mathbf{x}_i$ şi calculează termenii $(T_{k_1}, \ldots, T_{k_{n_i}}) = \mathbf{T}_i$.

Notăm $\mathcal{J}_i = \{j_1, \ldots, j_{n_i}\},\ \mathcal{K}_i = \{k_1, \ldots, k_{n_i}\}$. Presupunem

$$\cup_{i=1}^n \mathcal{J}_i = \cup_{i=1}^n \mathcal{K}_i = \{1, \ldots, n\},$$

şi

$$i \neq j \quad \Rightarrow \quad \begin{array}{l} \mathcal{J}_i \cap \mathcal{J}_j = \emptyset \\ \mathcal{K}_i \cap \mathcal{K}_j = \emptyset \end{array}.$$

Renumerotăm termenii şi variabilele luând

$$\begin{array}{rl} \mathcal{J}_1 = \mathcal{K}_1 = & \{1, \ldots, n_1\} \\ \mathcal{J}_2 = \mathcal{K}_2 = & \{n_1 + 1, \ldots, n_1 + n_2\} \\ \quad \vdots & \\ \mathcal{J}_L = \mathcal{K}_L = & \{n_1 + \ldots + n_{L-1}, \ldots, n_1 + \ldots + n_{L-1} + n_L = n\} \end{array}$$

Funcţia T se reprezintă prin $T(x) = (\mathbf{T}_i(x))_{1 \leq i \leq L},\ x = (\mathbf{x}_1, \ldots, \mathbf{x}_L)^T,\ \mathbf{x}_i \in \mathbb{R}^{n_i}$.

Un ciclu al unui algoritm paralel pentru calculul punctului fix utilizând şirul (12.1) presupune două faze:

- de calcul – procesul i calculează componenta $\mathbf{x}_i$
- de comunicaţie – procesul i trimite componenta calculată celorlalte procese şi recepţionează celelelte componente $\mathbf{x}_j,\ j \neq i$.

Faza de comunicaţie se poate programa - funcţie de suportul hard/soft utilizând

- memorie partajată,
- mesaje.

Există două variante de realizare a unui algoritm paralel pentru calculul punctului fix utilizând şirul (12.1):

- *sincron* – După efectuarea fazei de calcul are loc o sincronizare de tip barieră, iar apoi are loc faza de comunicaţie. Şirul $(x^k)_{k\in\mathbb{N}}$ generat coincide cu cel furnizat de un algoritm secvenţial. Faza de calcul a procesorului i constă din

$$\mathbf{x}_i^{k+1} = \mathbf{T}_i(\mathbf{x}_1^k, \ldots, \mathbf{x}_L^k)$$

- *asincron* – Se renunţă la sincronizarea barieră.

Rezultatulde convergenţă pentru o metodă bazată pe un algoritm paralel şi iterativ sincron coincide cu cel pentru metoda secvenţială.

12.1 Algoritm paralel şi iterativ asincron

Fiecare proces i poate contoriza într-un contor k_i iteraţiile pe care le execută. Odată calculată componenta $\mathbf{x}_i^{k_i+1}$ ea este trimisă celorlalte procese. Pentru calculul acestei componente procesul i utilizează pentru $\mathbf{x}_j,\ j \neq i$ valori care pot să nu coincidă cu ultimele valori calculate, respectiv de procesele $j \neq i$.

Notăm cu $s_j^i(k_i)$ indicele lui $\mathbf{x}_j$ furnizat de procesul j procesului i, valoare utilizată pentru calculul lui $\mathbf{x}_i^{k_i+1}$,

$$\mathbf{x}_i^{k_i+1} = \mathbf{T}_i(\mathbf{x}_1^{s_1^i(k_i)}, \ldots, \mathbf{x}_{i-1}^{s_{i-1}^i(k_i)}, \mathbf{x}_i^{k_i}, \mathbf{x}_{i+1}^{s_{i+1}^i(k_i)}, \ldots, \mathbf{x}_L^{s_L^i(k_i)}).$$

$\mathbf{x}_j^{s_j^i(k_i)}$ este utilizat de procesul i până la recepţia unei noi valori.

Pseudocodul procesului i este dat în Algoritmul 2. Algoritmul utilizează o metodă *convergence*():**boolean** care depistează convergenţa şi conţine o regulă de oprire. Operaţia **receive** este presupusă neblocantă.

Presupunem că

$$\lim_{k_i\to\infty} s_j^i(k_i) = \infty, \qquad \forall\ i, j \in \{1, \ldots, L\}. \tag{12.2}$$

Semnificaţia acestei restricţii este aceea că în lipsa regulii de oprire, toate procesele conlucrează, adică trimit şi recepţionează valori, niciun proces nu este lăsat deoparte.

Algorithm 2 Procesul i

```
1: procedure PROCES(i, x^0)
2:     k_i ← 0
3:     do
4:         receive x_j^{s_j^i(k_i)}, ∀ j ≠ i
5:         Compute x_i^{k_i+1}
6:         send x_i^{k_i+1}
7:         k_i ← k_i + 1
8:     while convergence()
9: end procedure
```

Considerăm $T : D \subseteq \mathbb{R}^n = \prod_{j=1}^{L} \mathbb{R}^{n_j} \to \prod_{j=1}^{L} \mathbb{R}^{n_j}$, şi notăm norma din $\mathbb{R}^{n_j}$ prin $\|\cdot\|_j$ iar norma din $\mathbb{R}^n$ va fi $\|x\| = \max_{1\le j\le L} \|\mathbf{x}_j\|_j$.

Presupunem că există $\alpha \in [0, 1)$ astfel încât

$$\|T(x) - T(x^*)\| \le \alpha\|x - x^*\|, \quad \forall\ x \in D. \tag{12.3}$$

În fiecare proces $i \in \{1, \ldots, L\}$ se pune în evidenţă subşirul $(m_l^i)_{l\in\mathbb{N}}$ prin

$$\begin{aligned} m_0^i &= 0 \\ m_{l+1}^i &= \min\{k_i : s_j^i(k_i) > m_l^i,\ \forall\ j \ne i\}. \end{aligned}$$

$\mathbf{x}_i^{m_{l+1}^i}$ se obţine după ce toate componenetele necesare calculului au fost reînnoite. Semnificaţia inegalităţii $s_j^i(k_i) > m_l^i$ constă în aceea că procesul i a recepţionat cel puţin o valoare nouă pentru $\mathbf{x}_j$, după aceea utilizată pentru calculul lui $x^{m_l^i}$.

Notăm $x^{m_{l+1}^i} = (\mathbf{x}_1^{s_1^i(k_i)}, \ldots, \mathbf{x}_{i-1}^{s_{i-1}^i(k_i)}, \mathbf{x}_i^{m_{l+1}^i}, \mathbf{x}_{i+1}^{s_{i+1}^i(k_i)}, \ldots, \mathbf{x}_L^{s_L^i(k_i)})^T$ cu $k_i = m_{l+1}^i - 1$.

Are loc următorul rezultat de convergenţă:

Teorema 12.1.1 *(El Tarazi) În ipotezele introduse, (12.2), (12.3), în fiecare proces* $\lim_{l\to\infty} x^{m_l^i} = x^*$.

Demonstaţie. Pentru $i \in \{1, \ldots, L\}$, $\mathbf{x}_i^1 = \mathbf{T}_i(x^0)$, de unde

$$\|\mathbf{x}_i^1 - \mathbf{x}_i^*\|_i = \|\mathbf{T}_i(x^0) - \mathbf{T}_i(x^*)\|_i \le \max_{1\le j\le L} \|\mathbf{T}_j(x^0) - \mathbf{T}_j(x^*)\|_j =$$

$$= \|T(x^0) - T(x^*)\| \le \alpha\|x^0 - x^*\|,$$

şi astfel $\|x^{m_1^i} - x^*\| \le \alpha\|x^0 - x^*\|$.

Arătăm că $\|x^{m_l^i} - x^*\| \le \alpha^l\|x^0 - x^*\|$.

După calculul elementului $x^{m_l^i}$, procesul i va recepţiona cel puţin o valoare pentru $\mathbf{x}_j$, adică

$$s_j^i(k_i) > m_l^i \quad \Rightarrow \quad \|\mathbf{x}_j^{s_j^i(k_i)} - \mathbf{x}_j^*\|_j \le \alpha^{l+1}\|x^0 - x^*\| \qquad \forall\ j \ne i.$$

Apoi $\|\mathbf{x}_i^{m_{l+1}^i} - \mathbf{x}_i^*\|_i =$

$$= \|\mathbf{T}_i((\mathbf{x}_1^{s_1^i(k_i)}, \ldots, \mathbf{x}_{i-1}^{s_{i-1}^i(k_i)}, \mathbf{x}_i^{k_i}, \mathbf{x}_{i+1}^{s_{i+1}^i(k_i)}, \ldots, \mathbf{x}_L^{s_L^i(k_i)})^T) - \mathbf{T}_i(x^*)\|_i \leq$$

$$\leq \|T((\mathbf{x}_1^{s_1^i(k_i)}, \ldots, \mathbf{x}_{i-1}^{s_{i-1}^i(k_i)}, \mathbf{x}_i^{k_i}, \mathbf{x}_{i+1}^{s_{i+1}^i(k_i)}, \ldots, \mathbf{x}_L^{s_L^i(k_i)})^T) - T(x^*)\| \leq$$

$$\leq \alpha\|(\mathbf{x}_1^{s_1^i(k_i)}, \ldots, \mathbf{x}_{i-1}^{s_{i-1}^i(k_i)}, \mathbf{x}_i^{k_i}, \mathbf{x}_{i+1}^{s_{i+1}^i(k_i)}, \ldots, \mathbf{x}_L^{s_L^i(k_i)})^T - x^*\| \leq$$

$$\leq \alpha^{l+1}\|x^0 - x^*\|.$$

Astfel

$$\|x^{m_{l+1}^i} - x^*\| = \max\{\|\mathbf{x}_i^{m_{l+1}^i} - \mathbf{x}_i^*\|_i, \max_{j \neq i} \|\mathbf{x}_j^{s_j^i(k_i)} - \mathbf{x}_j^*\|_j\} \leq \alpha^{l+1}\|x^0 - x^*\|.$$

■

Capitolul 13

OpenCL prin *Aparapi*

Procesoarele grafice (*Graphical Processing Unit - GPU*), unităţile de procesare (*multi-core Central Processing Unit*) sunt utilizate în calculul paralel.

În acest sens au fost dezvoltate instrumente de programare:

- *CUDA - Compute Unified Device Arhitecture* este o platformă şi model de programare pentru procesoarele grafice Nvidia. CUDA este dezvoltat de Nvidia.
- *OpenCL - Open Computing Language* este un standard de programare menţinut de organizaţia non-profit *Khronos Group* şi vizează utilizarea unităţilor de procesare (CPU) şi a procesoarelor grafice (GPU). Dezvoltat iniţial de *Apple* în prezent este adoptat de *AMD, Intel, Nvidia, ARM Holdings.*

  Modelul de programare paralelă este SIMD.

Fiecare instrument este reprezentat de biblioteci de funcţii iar limbajul de dezvoltare este C.

Intel dezvoltă arhitectura *Many-Integrated-Core* (MIC), destinat de asemenea calculului paralel.

13.1 *Aparapi*

*Aparap*i -(*A PARallel API*)[1] este un cadru de lucru care permite programarea în Java pentru platforma OpenCL - `https://github.com/aparapi/aparapi/releases`. Ideea cadrului de lucru este convertirea codului Java în *cod OpenCL* în timpul execuţiei.

În funcţie de resursele existente pe calculatorul de lucru execuţia aplicaţiei se va face utilizând GPU sau într-un bazin de fire de execuţie JTP *Java Thread Pool.*

Pe un calculator cu placă grafică AMD este posibil ca pentru utilizarea GPU să fie necesară instalarea produsului *AMD-APP-SDK-*.*

Aparapi se distribuie sub forma unei arhive care trebuie dezarhivată. Arhiva conţine fişiere `dll` prin care se interacţionează cu platforma OpenCL.

Compilarea unui program

[1] *apa rapi* în indoneziană înseamnă *ce îngrijit.*

```
set APARAPI_DIR=. . .
javac -g -cp %APARAPI_DIR%\aparapi.jar *.java
```

Semnificaţia opţiunii *-g* este generarea tuturor informaţiilor de depanare.

Lansarea în execuţie

```
java -Djava.library.path=%APARAPI_DIR% -cp %APARAPI_DIR%\aparapi.jar;. ClasaMain
```

Modul de procesare este redat în Fig. 13.1 (imagine preluată din *QuickReference.pdf*).

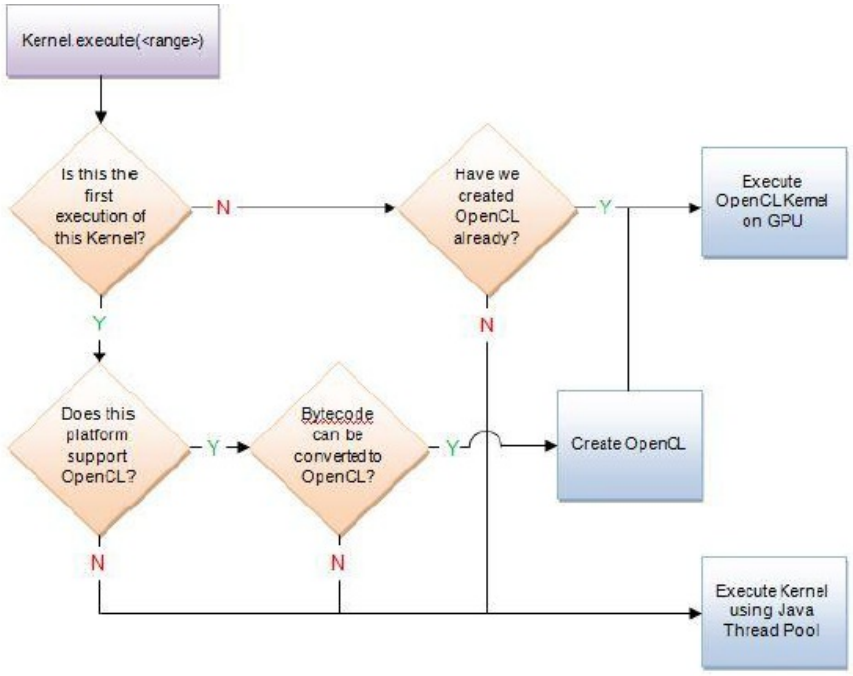


Fig. 13.1: Modul de procesare *aparapi*

13.2 Programare în *aparapi*

Şablonul unui program *aparapi* este

```
import com.amd.aparapi.Kernel;
import com.amd.aparapi.Range;

public class NumeClasa{

   public static void main(String[] _args) {
      . . .
      Kernel kernel = new Kernel(){
         @Override
          public void run() {
            . . .
         }
      };
      kernel.execute(Range.create(. . .));
      System.out.println("Execution mode=" + kernel.getExecutionMode());
```

```
      kernel.dispose();
      . . .
   }
}
```

Acţiunea din metoda `run` a obiectului *Kernel* conţine prelucrarea executată de o unitate de procesare sau fir de execuţie, funcţie de modul de lucru.

Datele cu care se fac calcule trebuie să fie de tip `int` sau `float` iar tablourile unidimensionale. Transformarea unei matrice $mat \in M_{r,c}(T)$ într-un vector $v \in T^{rc}$ prin juxtapunerea liniilor se face prin formula

$$v_{ic+j} = mat_{i,j}, \quad \forall (i,j) \in \{0,\ldots,r-1\} \times \{0,\ldots,c-1\},$$

unde $T \in \{$`int,float`$\}$.

Clasa `com.amd.aparapi.Kernel`

Constructori

- `Kernel()`

Metode

- `public abstract void run()`
- `public Kernel.EXECUTION_MODE getExecutionMode()`

  Evidenţiază modul lucru care poate fi GPU, JTP, CPU.
- `public Kernel execute(Range` *range*`)`
  `public Kernel execute(Range` *range,* `int` *n*`)`

  range indică structura unităţilor de procesare, iar n indică numărul de câte ori se execută mtoda `run` dintr-o unitate de procesare.

Clasa Kernel conţine o familie de funcţii matematica de argument şi valoare de tip `float,` dar care nu pot fi utilizate decăt în instanţierea clasei. Funcţiile înlocuiesc pe cele din clasa `java.lang.Math` care nu pot fi folosite.

Clasa `com.amd.aparapi.Range`

Specifică modul de organizare / gestiune a unităţilor de procesare. **Metode**

- `public static Range create(int` *globalWidth*`)`
- `public static Range create2D(int` *globalWidth,* `int` *globalHeight*`)`
- `public static Range create3D(int` *globalWidth,* `int` *globalHeight,*
  `int` *globalDepth,*`)`

13.3 Exemple

Tehnica de programare este ilustrată prin câteva exemple simple.

Produsul a două matrice

Date fiind $A \in M_{m,n}(\mathbb{Z}), B \in M_{n,p}(\mathbb{Z})$ se calculează $C = AB$. Componentele matricei C se calculează în paralel, fiecare componentă este calculată de o unitate de procesare GPU,

$$C_{i,j} = \sum_{k=0}^{n-1} A_{i,k}B_{k,j}, \quad \forall (i,j) \in \{0,\ldots,m-1\} \times \{0,\ldots,p-1\}.$$

Unităţile de procesare sunt organizate într-o matrice de dimensiune (m,p).

```
1  import com.amd.aparapi.Kernel;
2  import com.amd.aparapi.Range;

4  public class ProdMat{
5    public static void main(String[] args) {
6      final int m=3;
7      final int n=4;
8      final int p=5;
9      final int[] a=initA(m,n);
10     final int[] b=initB(n,p);
11     final int[] c=new int[m*p];

13     Kernel kernel = new Kernel(){
14       @Override public void run() {
15         int gidx = getGlobalId(0);
16         int gidy = getGlobalId(1);
17         int s=0;
18         for(int k=0;k<n;k++)
19           s+=a[gidx*n+k]*b[k*p+gidy];
20         c[gidx*p+gidy]=s;
21       }
22     };

24     long inceput=System.currentTimeMillis();
25     kernel.execute(Range.create2D(m,p));
26     long sfarsit=System.currentTimeMillis();

28     System.out.println("Execution mode=" + kernel.getExecutionMode());
29     kernel.dispose();
30     System.out.println("Durata : "+(sfarsit-inceput));

32     System.out.println("Matricea produs : ");
33     for (int i = 0; i < m; i++) {
34       for(int j=0;j<p;j++)
35         System.out.printf("%6d %1s", c[i*p+j]," ");
36       System.out.printf("\n");
37     }
38   }

40   /*
41      Transformarea unei matrice in vector
42      prin juxtapunerea liniilor
43   */
44   private static int[] iMat2vect(int[][] mat){
45     int rows=mat.length;
46     int cols=mat[0].length;
47     int[] v=new int[rows*cols];
```

```
48      for(int i=0;i<rows;i++){
49        for(int j=0;j<cols;j++){
50          v[i*cols+j]=mat[i][j];
51        }
52      }
53      return v;
54    }

56    /*
57       Initializarea primei matrice A
58    */
59    private static int[] initA(int m,int n){
60      int[][] mat=new int[m][n];
61      int fact=1;
62      for(int i=0;i<m;i++){
63        for(int j=0;j<n;j++){
64           mat[i][j]=fact*(j+1);
65        }
66        fact*=10;
67      }
68      System.out.println("Matricea A :");
69      for(int i=0;i<m;i++){
70        for(int j=0;j<n;j++) System.out.print(mat[i][j]+" ");
71        System.out.println();
72      }
73      return iMat2vect(mat);
74    }

76    /*
77       Initializarea celei de a doua matrice B
78    */
79    private static int[] initB(int m,int n){
80      int[][] mat=new int[m][n];
81      int fact=1;
82      for(int i=0;i<m;i++){
83        for(int j=0;j<n;j++){
84           mat[i][j]=(i+1)*(j+1);
85        }
86      }
87      System.out.println("Matricea B :");
88      for(int i=0;i<m;i++){
89        for(int j=0;j<n;j++) System.out.print(mat[i][j]+" ");
90        System.out.println();
91      }
92      return iMat2vect(mat);
93    }
94  }
```

În cod inițializarea este făcută în mod convenabil pentru verificarea facilă a rezultatelor.

Codul fişierului de comenzi al execuţiei este

```
1 set ProgName=ProdMat
2 del %ProgName%*.class
3 set APARAPI_DIR=. . .
4 javac -g -cp %APARAPI_DIR%\aparapi.jar %ProgName%.java
5 java -Djava.library.path=%APARAPI_DIR% -cp %APARAPI_DIR%\aparapi.jar;. %ProgName%
```

Calculul unei integrale prin metoda trapezelor

Fie $f : [a, b] \to \mathbb{R}$ o funcţie continuă. Pentru un număr $m \in \mathbb{N}^*$, metoda trapezelor constă din calculul expresiei $I_m(f, a, b)$, unde

$$\int_a^b f(x)\mathrm{d}x \approx I_m(f, a, b) = \frac{b-a}{2m}[f(a) + 2\sum_{i=1}^{m-1} f(a + ih) + f(b)].$$

Algoritmul care va fi utilizat în program foloseşte datele

- $size \in \mathbb{N}^*$;
- $\alpha_i = a + i\frac{b-a}{size},\ i \in \{0, 1, \ldots, size\}$;

şi calculează şirul $partial_i = I_m(f, \alpha_i, \alpha_{i+1})$. În final

$$\int_a^b f(x)\mathrm{d}x = \sum_{i=0}^{size-1} \int_{\alpha_i}^{\alpha_{i+1}} f(x)\mathrm{d}x \approx \sum_{i=0}^{size-1} partial_i.$$

Unităţile de procesare sunt organizate ca un vector de dimensiune *size*.

Codup aplicaţiei este

```
1  import com.amd.aparapi.Kernel;
2  import com.amd.aparapi.Range;
3
4  public class Integ{
5    public static void main(String[] _args) {
6      final int size=16;
7      final float[] data={0.0f,0.25f*(float)Math.PI};
8      final float[] partial=new float[size];
9
10     Kernel kernel = new Kernel(){
11       /*
12         Definirea functiei de integrat
13       */
14       float fct(float x){
15         return log(1+tan(x));
16       }
17
18       /*
19         Formula trapezelor aplicata functiei fct
20         in intervalul [a,b] cu parametrul de discretizare m
21       */
22       float trapeze(float a,float b,int m){
23         float s=0,h=(b-a)/m;
24         for(int i=1;i<m;i++) s+=fct(a+i*h);
25         return 0.5f*h*(fct(a)+2*s+fct(b));
26       }
27
28       @Override
29       public void run() {
30         int gid = getGlobalId();
31         float h=(data[1]-data[0])/size;
32         float a=data[0]+h*gid;
33         float b=a+h;
34         partial[gid]=trapeze(a,b,500);
35       }
36     };
```

```
38      kernel.execute(Range.create(size));
39      System.out.println("Execution mode=" + kernel.getExecutionMode());
40      kernel.dispose();

42      /*
43        Insumarea rezultatelor partiale
44      */
45      float integ=0;
46      for (int i = 0; i < size; i++) {
47         integ+=partial[i];
48      }

50      System.out.printf("Integrala = %10.4f\n", integ);
51    }
52  }
```

Rezolvarea sistemelor algebrice de ecuaţii liniare prin metoda Jacobi

$A \in M_n(\mathbb{Z}), b \in \mathbb{R}^n$. Pentru rezolvarea sistemului $Ax = b$ potrivit metodei Jacobi se construiesc şirurile

$$u_i^{(k+1)} = \frac{1}{a_{i,i}}(b_i - \sum_{j=1, j\neq i}^{n} a_{i,j}u_j^{(k)}), \quad i \in \{1, \ldots, n\},\ k \in \mathbb{N}.$$

Algoritmul va fi paralel şi iterativ asincron, numărul iteratiilor va fi fixat a priori iar in final se va determina un indicator de raspuns.

Problema de test este

$$\begin{pmatrix} 9 & -1 & 3 & 1 & 3 \\ 2 & -8 & 0 & 1 & 3 \\ -2 & 2 & 7 & 1 & 1 \\ 4 & -1 & -1 & 9 & 2 \\ -1 & -3 & 0 & 2 & 6 \end{pmatrix} x = \begin{pmatrix} -5 \\ -15 \\ 16 \\ 20 \\ 1 \end{pmatrix}$$

cu soluţia $x = (-1, 2, 1, 3, 0)^T$.

Aplicaţia are codul

```
1  import com.amd.aparapi.Kernel;
2  import com.amd.aparapi.Range;

4  public class JacobiAsync{
5    public static void main(String[] _args) {
6      final float[] a=initA();
7      final float[] b=initB();
8      final int n=b.length;
9      // Afisarea datelor sistemului
10     for(int i=0;i<n;i++){
11       for(int j=0;j<n;j++) System.out.print(a[i*n+j]+" ");
12       System.out.println(" <-> "+b[i]);
13     }

15     final float[] x=new float[n];
16     final float[] y=new float[n];
17     final float[] errors=new float[n];
18     float tol=1e-5f,nrm=0.0f;
```

```
19      int nmi=50;

21      Kernel kernel = new Kernel(){
22        @Override public void run() {
23          int gid = getGlobalId();
24          for(int i=0;i<n;i++)x[i]=y[i];
25          float s=0;
26          for(int i=0;i<n;i++){
27            if(i!=gid) s+=a[gid*n+i]*x[i];
28          }
29          y[gid]=(b[gid]-s)/a[gid*n+gid];
30          errors[gid]=Math.abs(y[gid]-x[gid]);
31        }
32      };

34      // Aproximatia initiala
35      for(int i=0;i<n;i++) y[i]=0.0f;
36      long inceput=System.currentTimeMillis();
37      kernel.execute(Range.create(n),nmi);
38      long sfarsit=System.currentTimeMillis();
39      System.out.println("Execution mode=" + kernel.getExecutionMode());
40      kernel.dispose();
41      System.out.println("Durata : "+(sfarsit-inceput));

43      System.out.println("Solutia :");
44      for (int i = 0; i < n; i++) {
45         System.out.printf("%10.4f\n", y[i]);
46      }
47      for(int i=0;i<n;i++)nrm=Math.max(nrm,errors[i]);
48      int ind=0;
49      if(nrm>=tol) ind=1;
50      System.out.println("Indicatorul de raspuns : "+ind);
51    }

53    /*
54       Transformarea unei matrice in vector
55       prin juxtapunerea liniilor
56    */
57    private static float[] fMat2vect(float[][] mat){. . .}

59     /*
60       Initializarea matricei A
61    */
62    private static float[] initA(){
63      float[][] mat={{9,-1,3,1,3},{2,-8,0,1,3},{-2,2,7,1,1},{4,-1,-1,9,2},{-1,-3,0,2,6}};
64      int n=mat.length;
65      return fMat2vect(mat);
66    }

68    /*
69       Initializarea vectorului b
70    */
71    private static float[] initB(){
72      float[] mat={-5,-15,16,20,1};
73      return mat;
74    }
75  }
```

Varianta paralelă şi iterativ sincronă are codul

```
1  import com.amd.aparapi.Kernel;
2  import com.amd.aparapi.Range;

4  public class JacobiSync{
5    public static void main(String[] _args) {
6      final float[] a=initA();
7      final float[] b=initB();
```

```
8       final int n=b.length;
9       // Afisarea datelor sistemului
10      for(int i=0;i<n;i++){
11        for(int j=0;j<n;j++) System.out.print(a[i*n+j]+" ");
12        System.out.println(" <-> "+b[i]);
13      }

15      final float[] x=new float[n];
16      final float[] y=new float[n];
17      final float[] errors=new float[n];
18      float tol=1e-5f,nrm=0.0f;
19      int nmi=50;

21      Kernel kernel = new Kernel(){
22        @Override public void run() {
23          int gid = getGlobalId();
24          for(int i=0;i<n;i++)x[i]=y[i];
25          float s=0;
26          for(int i=0;i<n;i++){
27            if(i!=gid) s+=a[gid*n+i]*x[i];
28          }
29          y[gid]=(b[gid]-s)/a[gid*n+gid];
30          errors[gid]=Math.abs(y[gid]-x[gid]);
31        }
32      };

34      // Aproximatia initiala
35      for(int i=0;i<n;i++) y[i]=0.0f;
36      long inceput=System.currentTimeMillis();

38      int iter=0;
39      do{
40        iter++;
41        kernel.execute(Range.create(n));
42        nrm=0.0f;
43        for(int i=0;i<n;i++) nrm=Math.max(nrm,errors[i]);
44        //System.out.println("Iter : "+iter+" Norma erorii : "+nrm);
45      }
46      while((tol<=nrm)&&(iter<nmi));
47      long sfarsit=System.currentTimeMillis();
48      System.out.println("Durata : "+(sfarsit-inceput));
49      System.out.println("Execution mode=" + kernel.getExecutionMode());
50      kernel.dispose();

52      System.out.println("Solutia :");
53      for (int i = 0; i < n; i++) {
54         System.out.printf("%10.4f\n", y[i]);
55      }
56      int ind=0;
57      if(nrm>=tol) ind=1;
58      System.out.println("Indicatorul de raspuns : "+ind);
59    }

61    /*
62       Transformarea unei matrice in vector
63       prin juxtapunerea liniilor
64    */
65    private static float[] fMat2vect(float[][] mat){. . .}

67    /*
68       Initializarea matricei A
69    */
70    private static float[] initA(){. . .}

72    /*
73       Initializarea vectorului b
74    */
```

```
75   private static float[] initB(){. . .}
76 }
```

Se constată că durata de execuţie a variantei sincrone este puţin mai mare decât a variantei asincrone.

Produsele informatice utilizate

Pe durata existenţei, produsele informatice evoluează prin versiunile pe care producătorii ni le pun la dispoziţie. Nu de puţine ori o versiune nouă nu este compatibilă cu versiunea anterioară, fapt care necesită adaptarea programelor *client*.

Lista următoare precizează versiunile produselor utilizate în lucrare, indicate în majoritatea cazurilor prin resursa de instalare.

Versiunile produselor informatice utilizate în lucrare		
No.	Produsul informatic	Resursa/versiunea
1	apache-ant	apache.ant-1.9.6-bin.tar.gz
2	apache-commons-fileupload	commons-fileupload-1.3.1-bin.tar.gz
3	apache-commons-math	commons-math3-3.6-bin.tar.gz
4	apache-karaf	apache-karaf-4.0.4.zip
5	apache-maven	apache-maven-3.3.9-bin.tar.gz
6	apache-tomcat	apache-tomcat-8.0.32.tar.gz
7	aparapi	dist_windows_x86_64.zip
8	felix	felix-framework-5.4.0.tar.gz
9	glassfish	glassfish-4.1.1.zip
10	Google Web Toolkit	gwt-2.7.0.zip
11	Google App Engine	appengine-java-sdk-1.3.2.zip
12	httpcomponents-client	httpcomponents-4.5.1.tar.gz
13	Jama	Jama-1.0.3.jar
14	Java	jdk-8u74-windows-x64.exe
15	Java Expression Parser (JEP)	jep-2.4.1-ext-1.1.1-gpl.zip
16	jersey	jersey-ri-2.22.1.zip
17	jfreechart	jfreechart-1.0.19.zip
18	junit	junit-4.12.zip
19	Maple	Maple18WindowsInstaller.exe
20	MathEclipse parser	matheclipse-parser-0.0.10.jar
21	Mathematica	Mathematica_9.0.1_WIN.exe
22	metro	metro-standalone-2.3.1.zip
23	NetBeans	netbeans-8.1-javaee-windows.exe
24	PtPlot	ptplot5.10.tar.gz
25	Scilab	scilab-5.5.2.exe
26	symja	COMMONS_MATH4_SYMJA.jar symja-2016-01-09.jar

No.	Produsul informatic	Resursa/versiunea
27	visad	visad.jar

Bibliografie

[1] ANISIU V., 2006, *Calcul simbolic cu Maple.* Ed. Presa Universitară Clujeană, Cluj-Napoca.

[2] BOIAN F.M., BOIAN R. F., 2004, *Tehnologii fundamentale Java pentru aplicaţii Web.* Ed. Albastră, Cluj-Napoca.

[3] KINCAID D., CHENEY W., 1991, *Numerical Analysis. Mathematics of scientific computing.* Brooks/Cole, Pacific Grove, California.

[4] LANDAU H. R., 2005, *A First Course in Scientific Computing. Symbolic, Graphic, and Numeric Modeling Using Maple, Java, Mathematica, and Fortran90.* Princeton Univ. Press, Princeton.

[5] MĂRUŞTER Şt., 1981, *Metode numerice în rezolvarea ecuaţiilor neliniare.* Ed. tehnică, Bucureşti.

[6] PETCU D., 2000, *Matematică asistată de calculator.* Ed. Eubeea, Timişoara.

[7] PRESS W. H., TEUKOLSKI S. A., VETTERING W. T., FLANNERY B. P., 2007, *Numerical Recipies 3rd Edition: The Art of Scientific Computation.* Cambridge University Press, Cambridge.

[8] RICHARSON L. J., 2003, *Visualizing Complex Functions.* `http://web.archive.org/web/20030802162645/physics.hallym.ac.kr/education/TIPTOP/VLAB/QmSct/complex.html`.

[9] SCHEIBER E., 2007, *Programare concurentă şi paralel-distribuită în Java.* Ed. Albastră, Cluj-Napoca.

[10] SCHEIBER E., 2010, *Java în calculul ştiinţific.* Ed. Universitii Transilvania Braşov.

[11] STANCU D. D., COMAN G., (Ed), 2001, *Analiză numerică şi teoria aproximării.* Vol. I, II, III, Ed. Presa Universitară Clujeană, Cluj-Napoca.

[12] TOCCI C., ADAMS S., 1996, *Applied Maple for Engineers and Scientist.* Artech House, Boston, London.

[13] ***, `http://en.wikipedia.org/wiki/List_of_numerical_analysis_software`.

[14] ***, http://en.wikipedia.org/wiki/Comparison_of_numerical_analysis_software.

Majoritatea produselor utilizate conţin documentaţie de utilizare, prezentarea tehnică a interfeţelor de programare (API), etc. La acestea se adaugă multe alte referinţe în Internet.