Laboratorul de **M**etode **N**umerice

Universitatea "POLITEHNICA" Bucureşti, Facultatea de Electrotehnică, http://lmn.pub.ro, lmn@lmn.pub.ro

Metode numerice în ingineria electrică

L0 - Preliminarii asupra Laboratorului de Metode Numerice

Autori:

Conf. Gabriela Ciuprina
Prep. Marius Piper
As. Marius Rădulescu
As. Mihai Popescu

25 februarie 2004

Cuprins

Capitolul 0

Preliminarii asupra *Laboratorului de Metode Numerice*

0.1 Informaţii utile

0.1.1 Regulament

Prezenţa nu este obligatorie. Atragem însă atenţia că laboratorul are 50 % din punctajul notei la această disciplină. Este aproape imposibil ca un student care are nota zero la laborator să promoveze această disciplină.

Deoarece posibilităţile de recuperare a laboratoarelor sunt limitate, refacerea unei lucrări de laborator se poate face în cursul săptămânii afectate lucrării şi numai în măsura în care există un calculator liber sau, în mod excepţional, în penultima săptămână de şcoală.

Predarea referatelor se face **personal**, în şedinţa următoare. Studenţii care se prezintă fără referat nu sunt primiţi la laborator. Studenţii care lipsesc nu trebuie să trimită referatele prin colegi.

0.1.2 Calendar

Săptămâna	Tema
1	Prezentare laborator
2	L1 - Algoritmi si structuri de date
3	L2 - Erori in calculele numerice
4	L3,4 - Metoda Gauss - fără şi cu pivotare
5	L8 - Metode iterative pentru rezolvarea sistemelor liniare
6	L10 - Rezolvarea circuitelor rezistive liniare
7	Test laborator I
8	L11 - Interpolarea
9	L14,15 - Derivarea şi integrarea numerică
10	L16 - Rezolvare ecuatii neliniare
11	L18 - Rezolvarea ecuatiilor diferentiale
12	L20 - Rezolvarea circuitelor în regim tranzitoriu
13	Refaceri lucrari de laborator
14	Test laborator II

0.1.3 Modul de notare al laboratorului

Fiecare temă de laborator este notată cu două note: o notă (între 1 şi 10) pentru referat şi o notă (care este 0, 5 sau 10) pentru implementarea unui algoritm din cei studiaţi în lucrarea respectivă.

Referatul va contine:

1. Numele studentului şi grupa din care face parte
2. Numele cadrului didactic îndrumator
3. Titlul lucrarii
4. Scopul lucrarii
5. Rezultate experimentale, grafice, etc. (conform cerinţelor din îndrumar)
6. Observaţii şi concluzii

Foarte important:

- Observaţiile şi, mai ales, concluziile au ponderea cea mai mare în nota primită pe referat. Un referat fără observaţii şi concluzii poate avea nota maximă 4 din 10.
- Referatul nu trebuie să conţină: descrierea lucrării, principiul algoritmilor, pseudocodul algoritmilor.
- Testele de laborator (din săptămânile 7 şi 14) vor consta în implementarea în C a unuia din pseudocodurile care nu au fost implementare în cadrul şedinţelor obişnuite. Spre deosebire de acestea, studenţii nu vor beneficia la aceste teste de ajutor din partea cadrului didactic îndrumător. Testele de laborator vor fi de asemenea notate cu 0, 5 sau 10.
- Calculul notei la laborator:

  Fie n numărul de teme efectuate. Punctajul la laborator (maxim 50) se calculează astfel:

$$\text{Punctaj laborator} = 3\frac{\text{suma note referate}}{n} + 2\frac{\text{suma note implementare si teste}}{n+2}.$$

0.1.4 Calculul notei finale la disciplina Metode Numerice

Examenul scris are două părţi. Examenul parţial se dă în timpul semestrului (o oră dintr-un curs) şi se poate reface la final. Fiecare din aceste examinări valorează 25 puncte.

Notă:

Atât examenul parţial cât şi cel final se dau **cu cărţile şi cursurile pe bancă**. Este interzis însă ca studenţii să schimbe între ei materialele documentare pe care le au.

Pe scurt: laborator = 50, partial teorie = 25, final teorie = 25. Remarcaţi că prezenţa fizică nu se punctează.

0.2 Modul de desfăşurare a unei şedinţe de laborator

Fiecare şedinţă de laborator durează 2 ore şi are o anumită tematică (vezi subcapitolul 0.1.2). Activitatea propriu-zisă pe care trebuie să o desfăşoare fiecare student are două părţi importante.

Prima parte constă în exploatarea unor programe demonstrative ce ilustrează tematica lucrării. Sistemul de operare sub care se lucrează este Linux (`http://www.linux.org`).

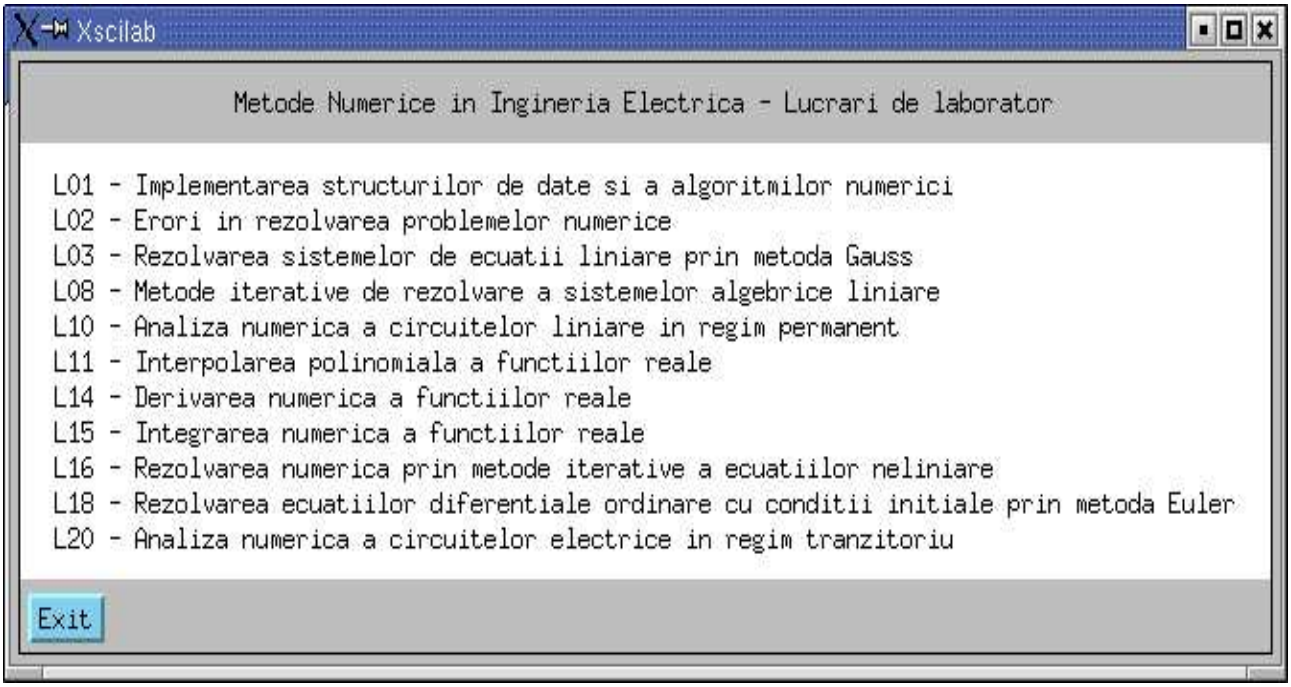


Figura 1: Meniul principal al programelor demonstrative.

Programele demonstrative sunt scrise in Scilab (`http://www-rocq.inria.fr/scilab/`) dar exploatarea lor nu necesită cunoaşterea acestui limbaj de programare. În urma exploatării acestor programe, studentul trebuie să redacteze un referat, conform cerinţelor fiecărei lucrări.

A doua parte a unei şedinţe de laborator constă în implementarea a cel puţin unuia din algoritmii studiaţi în cadrul temei respective. Implementarea se va face în limbajul C, în concordanţă cu pseudocodul prezentat în lucrare.

0.3 Programe demonstrative

Pentru lansarea programelor demonstrative faceţi un singur clic pe icoana cu sigla LMN. În continuare urmaţi modul de lucru descris în îndrumar.

Observaţie: programele demonstrative sunt disponibile la adresa `http://www.lmn.pub.ro/Education/methods/methods.html`. Pentru a putea lucra cu ele trebuie să aveţi instalat sistemul de operare Linux şi versiunea Scilab 2.7. Arhiva `LMN.tar.gz` trebuie decomprimată şi dezarhivată. Lansaţi Scilab şi la consola acestuia tastaţi `exec(main.sci)`. Meniul principal este cel prezentat în figura 1.

Notă importantă

Pentru ca un referat să fie luat în considerare, prezenţa la laborator este obligatorie. Este obligatoriu ca exploatarea acestor programe să fie făcută sub îndrumarea cadrului didactic.

Exerciţii:

1. Intraţi în contul dvs.;
2. Remarcaţi icoana LMN şi executaţi un singur clic pe ea;
3. Inchideţi programele demonstrative.

0.4 Implementare in C

Înainte de toate, vă este utilă o reîmprospătare a cunoştiinţelor de C dobândite în anii anteriori.

Metodele numerice ce vor fi studiate implementează în exclusivitate algoritmi numerici, care efectuează de cele mai multe ori calcule cu vectori şi matrice.

De aceea, în cele ce urmează vom face câteva consideraţii asupra declarării şi alocării vectorilor şi matricelor, astfel încât activitatea de programare în cadrul acestui laborator să fie cât mai eficientă.

Pentru început însă, vă recomandăm să studiaţi cu atenţie următoarele exemple şi apoi, pentru întelegerea lor, să citiţi paragrafele următoare.

0.4.1 Exemple

La adresa `http://www.lmn.pub.ro/Education/methods/methods.html` găsiţi două exemple pentru a vă familiariza cu stilul de lucru al acestui laborator.

1. Creaţi un director numit L0 cu comanda `mkdir L0`
2. Descarcaţi fişierele `nrutil_lmn.c, nrutil_lmn.h, aduna_vec.c, rw_matrix.c.`
3. Comentaţi conţinutul acestor fişiere.
4. Compilaţi primul exemplu cu comanda

```
gcc aduna_vec.c nrutil_lmn.c -o aduna_vec
```

5. Executaţi programul cu comanda

```
./aduna_vec
```

6. Compilaţi al doilea exemplu cu comanda

```
gcc rw_matrix.c nrutil_lmn.c -o rw_matrix
```

7. Executaţi programul cu comanda

```
./rw_matrix
```

Observaţie:

Comanda de compilare conţine un fişier sursă C (în care se află funţia `main` şi o funcţie ce implementează un anumit pseudocod), apoi fişierul `nrutil_lmn.c` (în care se află funcţiile de alocare/dealocare de memorie). Numele ce urmează după `-o` este numele programului executabil generat.

Iată conţinutul acestor fişiere:

Fişierul `aduna_vec.c`

```
#include "nrutil_lmn.h"

void aduna_vectori1 (int , VECTOR , VECTOR , VECTOR );

int
main (void)
{
/* program principal - adunarea a doi vectori
 * apeleaza aduna_vectori */

  int n; /* dimensiunea vectorilor */
  VECTOR a, b; /* vectorii de intrare */
  VECTOR c; /* vectorul rezultat c = a + b */

  int i;

  printf ("\n Introduceti dimensiunea vectorilor ");
  scanf ("%d", &n);

/* aloca spatiu de memorie pentru vectori */
  a = vector (1, n);
  b = vector (1, n);
  c = vector (1, n);

/* citeste vectorii a si b */
  for (i = 1; i <= n; i++)
    {
      printf ("\n a[%d] = ", i);
      scanf ("%f", &a[i]);
    }
  for (i = 1; i <= n; i++)
    {
      printf ("\n b[%d] = ", i);
```

```
      scanf ("%f", &b[i]);
    }

  aduna_vectori1 (n, a, b, c);

/* afiseaza rezultat */
  printf ("\n Rezultatul este \n");
  for (i = 1; i <= n; i++)
    printf ("c[%d] = %f \n", i, c[i]);


/* elibereaza spatiu de memorie */
  free_vector (a, 1, n);
  free_vector (b, 1, n);
  free_vector (c, 1, n);
  return (0);
}

void
aduna_vectori1 (int n, VECTOR a, VECTOR b, VECTOR c)
{
/* aduna doi vectori a + b, a caror indecsi incep de la 1 */
  int i;
  for (i = 1; i <= n; i++)
    c[i] = a[i] + b[i];
}
```

Fişierul `rw_matrix.c`:

```
#include "nrutil_lmn.h"

int
main (void)
{
/* program principal - citeste o matrice patrata, reala,  si o scrie */

  int n; /* dimensiunea matricelor */
  MATRIX a; /* matricea */

  int i, j;

  printf ("\n Introduceti dimensiunea matricei ");
  scanf ("%d", &n);

/* aloca spatiu de memorie pentru matrice  */
  a = matrix (1, n, 1, n);

/* citeste matricea */
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      {
printf ("\n a[%d][%d] = ", i, j);
scanf ("%f", &a[i][j]);
      }

/* afiseaza matricea  */
  printf ("\n Matricea citita este \n");
  for (i = 1; i <= n; i++)
    for (j = 1; j <= n; j++)
      printf ("a[%d][%d] = %f \n", i, j, a[i][j]);


/* elibereaza spatiu de memorie */
  free_matrix (a, 1, n, 1, n);
  return (0);
}
```

Fişierul `nrutil_lmn.c`

```
#include "nrutil_lmn.h"

void
nrerror (char error_text[])
```

```
{
  fprintf (stderr, "Run-time error...\n");
  fprintf (stderr, "%s\n", error_text);
  fprintf (stderr, "...now exiting to system...\n");
  exit (1);
}



VECTOR
vector (int nl, int nh)
{
  VECTOR v;

  v = (float *) malloc ((unsigned) (nh - nl + 1) * sizeof (float));
  if (!v)
    nrerror ("allocation failure in vector()");
  return v - nl;
}


IVECTOR
ivector (int nl, int nh)
{
  IVECTOR v;

  v = (int *) malloc ((unsigned) (nh - nl + 1) * sizeof (int));
  if (!v)
    nrerror ("allocation failure in ivector()");
  return v - nl;
}


MATRIX
matrix (int nrl, int nrh, int ncl, int nch)
{
  int i;
  MATRIX m;

  m = (float **) malloc ((unsigned) (nrh - nrl + 1) * sizeof (float *));
  if (!m)
    nrerror ("allocation failure 1 in matrix()");
  m -= nrl;

  for (i = nrl; i <= nrh; i++)
    {
      m[i] = (float *) malloc ((unsigned) (nch - ncl + 1) * sizeof (float));
      if (!m[i])
nrerror ("allocation failure 2 in matrix()");
      m[i] -= ncl;
    }
  return m;
}


IMATRIX
imatrix (int nrl, int nrh, int ncl, int nch)
{
  int i;
  IMATRIX m;

  m = (int **) malloc ((unsigned) (nrh - nrl + 1) * sizeof (int *));
  if (!m)
    nrerror ("allocation failure 1 in imatrix()");
  m -= nrl;

  for (i = nrl; i <= nrh; i++)
    {
      m[i] = (int *) malloc ((unsigned) (nch - ncl + 1) * sizeof (int));
      if (!m[i])
nrerror ("allocation failure 2 in imatrix()");
      m[i] -= ncl;
    }
  return m;
}

void
```

```
free_vector (VECTOR v, int nl, int nh)
{
  free ((char *) (v + nl));
}

void
free_ivector (IVECTOR v, int nl, int nh)
{
  free ((char *) (v + nl));
}

void
free_matrix (MATRIX m, int nrl, int nrh, int ncl, int nch)
{
  int i;

  for (i = nrh; i >= nrl; i--)
    free ((char *) (m[i] + ncl));
  free ((char *) (m + nrl));
}

void
free_imatrix (IMATRIX m, int nrl, int nrh, int ncl, int nch)
{
  int i;

  for (i = nrh; i >= nrl; i--)
    free ((char *) (m[i] + ncl));
  free ((char *) (m + nrl));
}
```

Fişierul `nrutil_lmn.h`

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<malloc.h>
#include<time.h>

typedef float **MATRIX;
typedef float *VECTOR;
typedef int **IMATRIX;
typedef int *IVECTOR;

void nrerror (char error_text[]);
VECTOR vector (int nl, int nh);
IVECTOR ivector (int nl, int nh);
MATRIX matrix (int nrl, int nrh, int ncl, int nch);
IMATRIX imatrix (int nrl, int nrh, int ncl, int nch);
void free_vector (VECTOR v, int nl, int nh);
void free_ivector (IVECTOR v, int nl, int nh);
void free_matrix (MATRIX m, int nrl, int nrh, int ncl, int nch);
void free_imatrix (IMATRIX m, int nrl, int nrh, int ncl, int nch);
```

0.4.2 Vectori

În C există o strânsă corespondenţă între adrese (pointeri) şi tablouri. În acest paragraf vom considera tablourile unidimensionale.

Valoarea reprezentată de `a[j]` este acelaşi lucru cu `*(a+j)` adică "conţinutul adresei obţinute incrementând pointer-ul `a` cu `j`. O consecinţă a acestei definiţii este aceea că dacă `a` este adresa unei locaţii valide, atunci `a[0]` este întotdeauna definit. Tablourile unidimenionale au în C, în mod natural, originea în 0. Un şir definit de

```
float b[4]
```

are referinţele valide `b[0]`, `b[1]`, `b[2]` şi `b[3]`, dar nu şi `b[4]`.

Problema este că mulţi algoritmi sunt descrişi în mod natural cu indici care încep de la 1. Cu siguranţă că aceşti algoritmi pot fi modificaţi, dar aceasta presupune o aritmetică suplimentară în operarea cu indici, lucru care nu este prea plăcut. Putem însă folosi puterea limbajului C pentru ca această problemă să dispară. Ideea este simplă:

```
float b[4], *bb;
bb = b - 1;
```

Pointer-ul `bb` indică acum o locaţie înaintea lui `b`. În consecinţă, elementele `bb[1]`, `bb[2]`, `bb[3]` şi `bb[4]` există şi vectorul `bb` are indici ce pornesc de la 1.

Uneori este convenabil să avem vectori care pornesc din 0, iar alteori este convenabil să avem vectori care pornesc din 1. De exemplu, coeficienţii unui polinom $a_0 + a_1 x + \ldots a_n x^n$ necesită un vector cu indici ce încep cu 0, pe când termenul liber al unui sistem de ecuaţii $b_i, i = 1, \ldots, n$ necesită un vector cu indici ce încep din 1.

Pentru a evita rescrierea algoritmilor ce sunt deduşi în mod natural cu indici ce pornesc de la 1, puteţi folosi o funcţie cu următoarea definiţie.

```
typedef float *VECTOR;

VECTOR vector (int nl, int nh)
{
  VECTOR v;

  v = (float *) malloc ((unsigned) (nh - nl + 1) * sizeof (float));
  if (!v)
    nrerror ("allocation failure in vector()");
  return v - nl;
}
```

Această funcţie alocă un vector de variable de tip `float`, care vor fi accesate cu indici cuprinşi între `nl` şi `nh`.

O utilizare tipică a acestei funcţii este

```
float *b;
b = vector(1,7);
```

Această funcţie precum şi funcţii similare ce alocă vectori de întregi se găsesc în fişierul `nrutil_lmn.c` pe care îl puteţi descărca de la adresa
`http://www.lmn.pub.ro/Education/methods/methods.html`.

Acest fişier conţine şi rutinele corespunzătoare de dealocare. De exemplu, pentru dealocarea memoriei ocupate de vectorul definit mai sus, instrucţiunea este

```
free_vector(b,1,7);
```

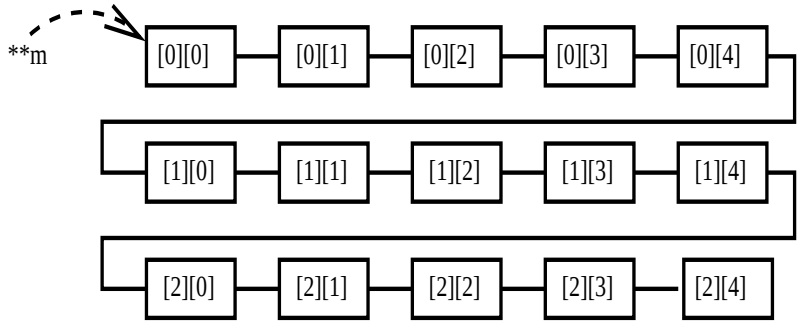
0.4.3 Matrice

Problema indicilor ce pornesc de la 0 sau de la 1 apare şi în cazul matricelor. În sintaxa C, lucrul cu tabele bidimensionale este puţin mai complicat. Să considerăm o valoare reală `a[i][j]` unde `i` şi `j` sunt întregi. Un compilator de C va genera coduri maşină diferite pentru această referinţă, aceasta depinzând de declaraţia pentru variabila `a`. Dacă `a` a fost declarată de dimensiune fixă, de exemplu `float a[2][4]` atunci codul maşină ar putea fi descris astfel: "la adresa `a` adună de 4 ori `i`, apoi adună `j` şi întoarce valoarea astfel adresată. Observaţi că valoarea constantă 4 trebuie cunoscută pentru a efectua în mod corect calculele.

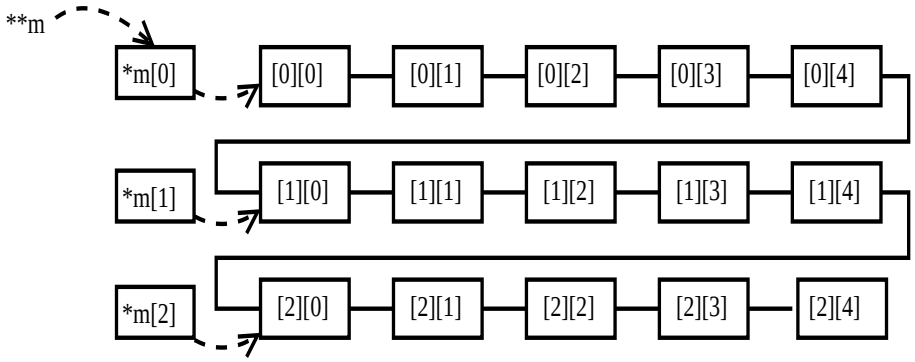
Să presupunem că `a` a fost declarat ca `float **a`. Atunci codul maşină `a[i][j]` este "la adresa `a` adună `i`, valoarea astfel adresată consider-o o nouă adresă, la care adună `j` şi întoarce valoarea astfel adresată.

Ilustrarea celor două moduri de accesare a unei valori este dată în figura 2. Observaţi că în al doilea caz nu este necesară cunoaşterea dimensiunii matricei şi că nu este nevoie de nici o înmulţire. O indirectare suplimentară înlocuieşte aceste informaţii. Această a doua schemă o recomandăm pentru lucrul la laboratorul de metode numerice.

Ideea pe scurt: vom evita tablourile bidimensionale de dimensiune fixă. Ele nu sunt structuri de date potrivite pentru reprezentarea matricelor în calculele ştiinţifice.



(a)



(b)

Figura 2: Două scheme de memorare pentru matricea `m`. Liniile întrerupte reprezintă pointeri la adrese, iar liniile continue conectează locaţii de memorie secvenţiale. (a) Pointer la un tablou bidimensional de dimensiune fixată; (b) Pointer la un tablou de pointeri către pointeri la linii.