

```

/*=====*\
filename: Gauss_Jordan.mu
-----
scopo: routine per ridurre per righe una matrice
      (ogni passaggio intermedio è mostrato e
      commentato)
-----
nota: la routine è pensata unicamente per uso
      didattico e potrebbe non funzionare corret-
      tamente con matrici simboliche. Per scopi
      non didattici usare la funzione di MuPAD

      linal::gauss_Jordan(M)
-----
autore: Claudio Marsan
        Liceo cantonale di Mendrisio
        Via Agostino Maspoli
        CH-6850 Mendrisio
        claudio.marsan@liceomendrisio.ch
-----
ultima modifica: 01.12.2005
-----
testato con: MuPAD Pro 3.1.1 for Windows
-----
OS: Microsoft Windows XP Professional sp2
-----
uso: read("../\Gauss_Jordan.mu"):
/*=====*\

```

```

// Carichiamo la libreria "linalg"
export(linalg):

```

```

// Stampa in un formato gradevole
PRETTYPRINT := TRUE:

```

```

// Gli elementi delle righe delle matrici sono separati da spazi bianchi
Pref::matrixSeparator("  "):

```

```

/*-----*\
| La procedura per ridurre per righe una matrice M |
\*-----*\
RMR := proc(M)
  local m, n, num_max_pivot, separazione, temp_j, num_riga,
        i, j, k, lista_pivot, num_pivot, s, t;
begin
  m := nrows(M);
  n := ncols(M);
  num_max_pivot := max(m, n);
  separazione := "=====";

  temp_j := 0;
  lista_pivot := [];

  print(Unquoted, "La matrice da ridurre per righe");
  print(M);
  print(Unquoted, separazione);

  // Trasformazione in matrice a gradini
  for num_riga from 1 to num_max_pivot do

    // Ricerca della colonna-pivot
    j := temp_j;
    repeat
      j := j + 1;
      until (nonZeros(col(M[num_riga..m, j..n], 1)) > 0)
    end_repeat;

    temp_j := j;

    if (j <= n) then

      // Ricerca del pivot

```

```

i := num_riga - 1;
repeat
  i := i + 1;
  until ((M[i, j] <> 0) or (i = m))
end_repeat;

lista_pivot := [[num_riga, j]] . lista_pivot;

// Se il candidato pivot è zero scambiamo due righe
if (i <> num_riga) then
  print(Unquoted, "Scambio la riga " . expr2text(num_riga) .
    " con la riga " . expr2text(i));
  M := swapRow(M, num_riga, i);
  print(M);
  print(Unquoted, separazione);
end_if;

// Se il pivot è diverso da 1 dividiamo ogni elemento della riga per il pivot
if (M[num_riga, j] <> 1) then
  print(Unquoted, "Divido la riga " . expr2text(num_riga) .
    " per " . expr2text(M[num_riga, j]));
  M := multRow(M, num_riga, 1/M[num_riga, j]);
  print(M);
  print(Unquoted, separazione);
end_if;

// Inserimento degli zeri nella colonna-pivot al di sotto del pivot
k := num_riga + 1;
while(k <= m) do
  if M[k, j] <> 0 then
    print(Unquoted, "Aggiungo alla riga " . expr2text(k) .
      " la riga " . expr2text(num_riga) .
      " moltiplicata per " . expr2text(-M[k, j]));
    M := addRow(M, num_riga, k, -M[k, j]);
    print(M);
    print(Unquoted, separazione);
  end_if;
  k := k + 1;
end_while;

end_if;

end_for;

// Riduzione per righe
num_pivot := nops(lista_pivot);
if (num_pivot > 1) then
  print(Unquoted, "La matrice e' a gradini e tutti i pivot valgono 1.");
  print(Unquoted, "Ora la riduco per righe.");
  print(Unquoted, separazione);
  for i from 1 to (num_pivot - 1) do
    s := lista_pivot[i][1];
    t := lista_pivot[i][2];
    for j from (s - 1) downto 1 do
      if (M[j, t] <> 0) then
        print(Unquoted, "Aggiungo alla riga " . expr2text(j) .
          " la riga " . expr2text(s) .
          " moltiplicata per " . expr2text(-M[j, t]));
        M := addRow(M, s, j, -M[j, t]);
        print(M);
        print(Unquoted, separazione);
      end_if;
    end_for;
  end_for;
end_if;
print(Unquoted, "La matrice e' ridotta per righe.");
end_proc:

```