noro_matrix

noro_matrix User's Manual
Edition 1.0
May 2008

by Masayuki Noro

1 行列演算パッケージ noro_matrix.rr

このマニュアルでは, asir-contrib パッケージに収録されている, 行列演算パッケージ'noro_matrix.rr' について解説する. このパッケージを使うには, まず'noro_matrix.rr' をロードする.

```
[1831] load("noro_matrix.rr");
[2014]
```

このパッケージの函数を呼び出すには, 全てlinalg. を先頭につける.

```
[2014] linalg.random_mat(3,5);
[ 0 -1 -1 ]
[ -1 3 0 ]
[ -2 -2 4 ]
```

このマニュアルでは, 関連する組込み函数についても解説する.

1.1 行列に関する函数

1.1.1 matrix,vector,linalg.unit_mat

matrix(*m*,*n*[,*listoflist*]) :: *m*行*n*列の行列を生成する.
vector(*size*[,*list*]) :: サイズが*size* のベクトルを生成する.
linalg.unit_mat(*size*) :: サイズが*size* の単位行列を生成する.

return 行列

size

m

n 正整数

listoflist リストのリスト

list リスト

- matrix, vector は組込み, linalg.unit_mat は'noro_matrix.rr' で定義されている.
- matrix, vector は, *listoflist*, *list* がない場合には零行列, 零ベクトルを生成する.
- *listoflist* は*[[1,2,3],[3,4,5]]* のようにリストからなるリストである. これが引数として与えられた場合, 要素であるリストを使って行列の各行が順に初期化される.
- *list* は*[1,2,3]* のようなリストである. これが引数として与えられた場合, このリストの要素によりベクトルの各成分が初期化される.

```
[1559] matrix(2,3);
[ 0 0 0 ]
[ 0 0 0 ]
[1560] vector(3);
[ 0 0 0 ]
[1561] linalg.unit_mat(3);
[ 1 0 0 ]
[ 0 1 0 ]
[ 0 0 1 ]
[1559] matrix(2,3,[[1,2,3],[4,5,6]]);
[ 1 2 3 ]
[ 4 5 6 ]
```

1.1.2 linalg.random_mat, linalg.random_rmat, linalg.random_vect

`linalg.random_mat(size,bound)` :: 正方整数行列をランダム生成する.
`linalg.random_rmat(m,n,bound)` :: *m* 行*n* 列の整数行列をランダム生成する.
`linalg.random_vect(size,bound)` :: 整数ベクトルをランダム生成する.

return 整数

number 整数

- linalg.random_mat(*size*,*bound*) は, サイズ*size*, 要素が*bound*未満の正方整数行列をランダム生成する.
-
- linalg.random_rmat(*m*,*n*,*bound*) は*m* 行*n* 列の, 要素が*bound*未満の整数行列をランダム生成する.
- linalg.random_vect(*size*,*bound*) は長さ*size*の, 要素が*bound*未満の整数ベクトルをランダム生成する.

```
[1579] linalg.random_mat(3,4);
[ 2 1 -2 ]
[ 0 -2 1 ]
[ 3 1 -2 ]
[1580] linalg.random_rmat(3,5,2);
[ 0 -1 0 0 0 ]
[ 0 -1 0 1 0 ]
[ -1 0 0 -1 1 ]
[1581] linalg.random_vect(3,6);
[ -3 2 3 ]
```

1.1.3 invmat

`invmat(mat)`
:: *mat* の逆行列を計算する.

return リスト

mat 正方行列

- 正方行列*mat* の逆行列を計算する.
- 結果は*[invmat,denom]* なるリストである. ここで, *invmat* は行列, *denom* は分母を表す式であり, *invmat/denom* が逆行列を表す.
- *mat*が整数行列, あるいは多項式行列の場合, *invmat* はそれぞれ整数行列, あるいは多項式となる. この仕様は, 無駄な分数, 有理式計算を省くために定められているが, 使いにくい場合もある.

```
[1575] A=linalg.random_mat(4,5);
[ 2 4 3 3 ]
[ 3 0 0 0 ]
[ 0 2 3 -2 ]
[ 2 0 -4 3 ]
[1576] L=invmat(A);
[[ 0 38 0 0 ]
```

```
[ -3 -28 63 45 ]
[ 18 16 -36 -42 ]
[ 24 -4 -48 -18 ],114]
[1577] AI=L[0]/L[1]$ AI*A;
[1578] [ 1 0 0 0 ]
[ 0 1 0 0 ]
[ 0 0 1 0 ]
[ 0 0 0 1 ]
```

1.1.4 det,nd_det

`det(`*mat*`[,`*mod*`])`
`nd_det(`*mat*`[,`*mod*`])`
:: *mat* の行列式を求める.

return 式

mat 行列

mod 素数

- `det` および`nd_det` は行列*mat* の行列式を求める.
- 引数*mod* がある時, GF(*mod*) 上での行列式を求める.
- 分数なしのガウス消去法によっているため, 多変数多項式を成分とする行列に対しては小行列式展開による方法のほうが効率がよい場合もある.
- `nd_det` は有理数または有限体上の多項式行列の行列式計算専用である. アルゴリズムはやはり分数なしのガウス消去法だが, データ構造および乗除算の工夫により, 一般に`det` より高速に計算できる.

```
[91] A=matrix(5,5)$
[92] V=[x,y,z,u,v];
[x,y,z,u,v]
[93] for(I=0;I<5;I++)for(J=0,B=A[I],W=V[I];J<5;J++)B[J]=W^J;
[94] A;
[ 1 x x^2 x^3 x^4 ]
[ 1 y y^2 y^3 y^4 ]
[ 1 z z^2 z^3 z^4 ]
[ 1 u u^2 u^3 u^4 ]
[ 1 v v^2 v^3 v^4 ]
[95] fctr(det(A));
[[1,1],[u-v,1],[-z+v,1],[-z+u,1],[-y+u,1],[y-v,1],[-y+z,1],[-x+u,1],
[-x+z,1],[-x+v,1],[-x+y,1]]
```

1.1.5 generic_gauss_elim

`generic_gauss_elim(`*mat*`)`
:: 整数行列を簡約する.

return リスト

mat 整数行列

- 整数行列*mat* の簡約階段形(reduced row echelon form; rref) を計算し, それを構成するデータをリストとして返す.
- 出力は*[B,D,J,K]* の形のリストである. 入力*mat* が*m* 行*n* 列とし, そのランクが*r* とすれば, *B* は*r* 行*n-r* 列の行列である. *D* は整数, *J* は長さ*r*, *K* は長さ*n-r* の整数ベクトルである.
- 出力データは*mat* の rref をエンコードしている. rref の第*J[l]* 列は*l* 行目のみが*D*, それ以外は 0 の列ベクトル, rref の第*K[l]* 列は*B* の第*l* 列を, 上から詰めたものとなる.
- このような形式で出力する理由は, 入力行列のランクが大きい場合に, rref は 0 が多い疎な行列となり, メモリを多く消費することと, この形の方が, プログラム上でデータを利用しやすいことによる.

```
[1600] A=linalg.random_rmat(3,5,2);
[ 0 -1 -1 0 -1 ]
[ 1 0 1 -1 0 ]
[ 1 1 0 0 0 ]
[1601] L=generic_gauss_elim(A);
[[ -1 -1 ]
[ 1 1 ]
[ -1 1 ],2,[ 0 1 2 ],[ 3 4 ]]
```

例えば, rref を計算する函数は次のように書ける.

```
def my_rref(A)
{
    S = size(A); M = S[0]; N = S[1];
    L = generic_gauss_elim(A);
    B = L[0]; D = L[1]; J = L[2]; K = L[3];
    R = length(J); NR = N-R;
    A1 = matrix(M,N);
    for ( I = 0; I < R; I++ ) {
        A1[I][J[I]] = D;
        for ( L = 0; L < NR; L++ ) A1[I][K[L]] = B[I][L];
    }
    return A1;
}
```

参照 Section 1.1.1 [matrix vector linalg.unit_mat], p. 1

1.1.6 linalg.compute_kernel, linalg.compute_image

`linalg.compute_kernel(`*mat*`[|rhs=`*vect*`])`
:: 有理数行列の核の基底を計算する.

`linalg.compute_image(`*mat*`)`
:: 有理数行列の像の基底を計算する.

return リスト

mat 有理数行列

vect 有理数ベクトル

- *m* 行*n* 列の行列を, 列ベクトルに左から掛けることにより*n* 次元ベクトル空間から*m* 次元ベクトル空間への線形写像とみなす.

- `linalg.compute_kernel` は有理数行列*mat* の核の基底を計算する.
- `linalg.compute_kernel` の出力は*[[v1,pos1],. . .,[vl,posl]]* の形のリストである. ここで, *vi* は基底ベクトル, *posi* は, *vi* の主成分位置, すなわち最小のインデックスを持つ成分の位置を表す. *posi* は全て異なることが保証される.
- オプション*vect* が指定された場合, 結果は*[sol,[[v1,pos1],. . .,[vl,posl]]]* なるリストとなる. ここで*sol* は*mat sol* = *vect* を満たすベクトル(特殊解), のこりは核の基底である.
- 解が存在しないような*vect* を指定するとエラーを起こす.
- `linalg.compute_image` は有理数行列*mat* の像の基底を計算する.
- `linalg.compute_image` の出力は, *[v1,pos1,hist1],. . .,[vl,posl,histl]* の形のリストである. ここで, *vi* は基底ベクトル, *posi* は, *vi* の主成分位置, すなわち最小のインデックスを持つ成分の位置を表す. *posi* は全て異なることが保証される. *histi* は, *vi* が, *mat* の列からどのように作られるかを示すデータである. 分散多項式で表現されており, 指数が行インデックス, 係数が, 一次結合の係数を表す. このデータにより作られるベクトルは, 定数倍を除いて*vi* に等しい.

```
[1643] A=linalg.random_rmat(3,5,3);
[ 2 1 0 1 -1 ]
[ 2 -2 1 0 1 ]
[ 2 1 -1 -1 -1 ]
[1644] linalg.compute_kernel(A);
[[[ 1 0 -8 4 6 ],0],[[ 0 1 2 -1 0 ],1]]
[1645] linalg.compute_kernel(A|rhs=vector(3,[1,2,3]));
[[ 0 0 8 -5 -6 ],[[[ 1 0 -8 4 6 ],0],[[ 0 1 2 -1 0 ],1]]]
[1646] linalg.compute_image(A);
[[[ 1 1 1 ],0,(1)*<<0>>],[[ 0 -3 0 ],1,(1)*<<1>>+(-1)*<<0>>],
[[ 0 0 3 ],2,(-3)*<<2>>+(-1)*<<1>>+(1)*<<0>>]]
```

1.1.7 `linalg.minipoly_mat`

`linalg.minipoly_mat(`*mat*`)`
:: 有理数行列*mat* の最小多項式を計算する.

return 一変数多項式

mat 有理数行列

- 有理数行列*mat* の最小多項式を計算し, 変数*x* の一変数多項式として返す.

```
[1682] A=linalg.random_mat(3,3);
[ -2 2 -2 ]
[ 0 1 -1 ]
[ 1 -2 -1 ]
[1683] linalg.minipoly_mat(A);
x^3+2*x^2-x-6
[1684] A^3+2*A^2-A-6*linalg.unit_mat(3);
[ 0 0 0 ]
[ 0 0 0 ]
[ 0 0 0 ]
```

1.1.8 linalg.jordan_canonical_form,linalg.sample_mat

`linalg.jordan_canonical_form(mat)`
:: 有理数正方行列のジョルダン標準形を計算する.

`linalg.sample_mat(list)`
:: 指定されたジョルダン標準形を持つ有理数正方行列を生成する.

return リスト

mat 有理数正方行列

list ジョルダンブロックのリスト

- `linalg.jordan_canonical_form(mat)` は有理数正方行列*mat* のジョルダン標準形を計算する.
- 出力は*[P,[[e1,s1,n1],. . .,[el,sl,nl]],defideal]* という形のリストである. ここで, *P* は変換行列, すなわち*P^(-1)AP* がジョルダン標準形となる正則行列, *[ei,si,ni]* は, 固有値*ei*, サイズ*si* のジョルダンブロックが*ni* 個並ぶことを意味する.
- 一般に, 出力は*a0*, . . . , *am* の形のパラメタを含む. これらは実際には, ある有理数体上既約な多項式の根である. これらを定義する方程式として, *defideal* が与えられる. *deflideal* はリストのリストであり, 各要素であるリストは, 一組の共役な根全体を定義するイデアルを表す. 実際には, 対応する一変数多項式の根を, 根と係数の関係により表したものである.
- `linalg.sample_mat(list)` は指定されたジョルダン標準形を持つような行列を生成する. *list* は*[[e1,s1],. . .,[el,sl]]* の形のリストで, *[ei,s1]* は固有値*ei*, サイズ*si* のジョルダンブロックを表す. 同じサイズのジョルダンブロックはいくつあってもよい.

```
[ 1 -2 0 ]
[ -1 2 1 ]
[ 0 -2 1 ]
[1807] L=linalg.jordan_canonical_form(A);
[[ 2 -2 0 ]
[ -1 0 1 ]
[ 2 -2 -1 ],[[2,1,1],[1,2,1]],[]]
[1808] P=L[0]$T=invmat(P)$PI=T[0]/T[1]$
[1809] [1810] [1811] PI*A*P;
[ 2 0 0 ]
[ 0 1 1 ]
[ 0 0 1 ]
[1810] A=linalg.sample_mat([[1,2],[1,1],[2,3],[2,1],[2,1]]);
[ 2 0 2 113 14 678 0 0 ]
[ -1 1 -2 -120 -14 -720 0 0 ]
[ -7 0 -13 -840 -105 -5040 0 0 ]
[ 54 54 0 380 0 2268 -54 0 ]
[ 1 0 2 112 16 672 0 0 ]
[ -9 -9 0 -63 0 -376 9 0 ]
[ 1 1 0 7 0 42 1 0 ]
[ 1 1 0 7 0 42 0 2 ]
[1811] L=linalg.jordan_canonical_form(A);
[[ 0 -6 42 0 0 0 2 2 ]
```

```
[ 0 6 0 0 0 0 0 -2 ]
[ 42 -294 0 0 0 0 -1 -1 ]
[ 0 0 0 6 0 108 0 0 ]
[ -6 42 48 0 0 0 0 0 ]
[ 0 0 -1 -1 0 -18 0 0 ]
[ 0 0 0 0 0 2 0 0 ]
[ 0 0 0 0 1 0 -2 0 ],[[2,3,1],[2,1,2],[1,2,1],[1,1,1]],[]]
```

参照 Section 1.1.7 [linalg.minipoly_mat], p. 5

Index

（インデックスがありません）

（インデックスがありません）

簡単な目次

1　行列演算パッケージ noro_matrix.rr 1
Index 8

目次

1 行列演算パッケージ noro_matrix.rr 1
1.1 行列に関する函数 1
1.1.1 matrix,vector,linalg.unit_mat 1
1.1.2 linalg.random_mat, linalg.random_rmat, linalg.random_vect 2
1.1.3 invmat 2
1.1.4 det,nd_det 3
1.1.5 generic_gauss_elim 3
1.1.6 linalg.compute_kernel, linalg.compute_image 4
1.1.7 linalg.minipoly_mat 5
1.1.8 linalg.jordan_canonical_form,linalg.sample_mat 6

Index 8