Tutorial 1

Betti numbers and generic ideals

How to create random inputs?

```
F := DensePoly(2);  -- dense poly of degree 2
Print "F is ", F;
Randomized(F);      -- does not modify F, returns a random poly
Print "F is ", F;
Randomize(F);       -- modifies F (does not return a value!)
Print "F is ", F;
```

Sometimes such "big" values are too big and lead to very expensive computations, so we want to control the range:

```
Rand(10, 20);  -- a random integer between 10 and 20

Define RandDensePoly(D, A, B)  -- a DensePoly with coeffs in A..B
  F := DensePoly(D);
  Return Sum([ Rand(A, B)*T | T In Support(F)]);
EndDefine;
RandDensePoly(2, -5, 5);

Define RandMat(R,C, A,B)  -- a RxC matrix with entries in A..B
  Return Mat([ [Rand(A,B) | J In 1..C] | I In 1..R]);
EndDefine;
RandMat(4,2, -2,2);
```

A compact way to write LDU decomposition

```
Define DSubMat(X, A, B)
  Return Det(Submat(X, A, B));
EndDefine;

N := 4;
Use Q[x[1..N, 1..N]];
X := MakeMatByRows(N, N, Indets());  -- (needs 4.7.2)

--define D
PM := [DSubMat(X, 1..K, 1..K) | K In 1..N];
D := DiagMat(Concat([PM[1]], [PM[I]/PM[I-1] | I In 2..N]));

--define L
Define LEntry(M,PM,I,J)
  If I=J Then Return 1; EndIf;
  If I<J Then Return 0; EndIf;
  Return DSubMat(M, Concat(1..(J-1),[I]), 1..J)/PM[J];
EndDefine;
L := Mat([[LEntry(X,PM,I,J) | J In 1..N] | I In 1..N]);

--define U
Define UEntry(M,PM,I,J)
  If I=J Then Return 1; EndIf;
  If I>J Then Return 0; EndIf;
  Return DSubMat(M, 1..I, Concat(1..(I-1),[J]))/PM[I];
EndDefine;
U := Mat([[UEntry(X,PM,I,J) | J In 1..N] | I In 1..N]);

L*D*U = X;
```