```
// Intrinsic functions.
//
// Quadratic form operations.
//    MinkowskiReduction, NormEchelon, InnerProduct.
// Predicates:
//    IsReduced, IsLessThan.
// Matrix operations.
//    ClassicalAdjoint, MinorDeterminant.

forward PositiveReduction, PermutationReduction, NeighborReduction;

//////////////////////////////////////////////////////////////////
//                                                              //
//                      QUADRATIC MODULES                       //
//                                                              //
//////////////////////////////////////////////////////////////////


intrinsic MinkowskiReduction(QF::AlgMatElt) -> AlgMatElt
   {Returns the unique Minkowski reduced representative
   for QF, and the transforming matrix. }

   RMat := Parent(QF);
   I := Identity(RMat);
   dim := Degree(RMat);
   Q0 := QF;
   QF, U := NormEchelon(QF);
   for k in [1..dim-1] do
      for j in [k+1..dim] do
         if QF[k,k] lt 2*AbsoluteValue(QF[j,k]) then
            E := I;
            E[j,k] := Sign(QF[j,k])*
               ( -AbsoluteValue(QF[j,k]) div QF[k,k] );
            QF, P := NormEchelon(E*QF*Transpose(E));
            U := P*E*U;
         end if;
      end for;
   end for;
   if IsReduced(QF) eq false then
      QF, U0 := $$(QF); U := U0*U;
   end if;
   /* Now look for local minimal forms. */
   U0 := Zero(Parent(U));
   while U0 ne I do
      QF, U1 := PermutationReduction(QF);
      QF, U2 := PositiveReduction(QF);
      QF, U3 := NeighborReduction(QF);
      U0 := U3*U2*U1; U := U0*U;
   end while;
   return QF, U;
end intrinsic;

intrinsic NormEchelon(QF::AlgMatElt) -> AlgMatElt
   {Returns the matrix obtained by a basis permutation
   such that QF[i,i] le QF[j,j] for all i le j. }

   RMat := Parent(QF);
   I := Identity(RMat);
   dim := Degree(RMat);
   U0 := I; U1 := I;
   for i in [1..dim-1] do
      if QF[i+1,i+1] lt QF[i,i] then
         P := I;
         P[i+1,i+1] := 0; P[i,i] := 0;
         P[i,i+1] := 1; P[i+1,i] := 1;
```

```
         QF := P*QF*P; U0 := P*U0;
      end if;
   end for;
   if U0 ne I then
      QF, U1 := $$(QF); // Recursion.
   end if;
   return QF, U1*U0;
end intrinsic;

intrinsic IsReduced(QF::AlgMatElt) -> BoolElt
   {Returns true if and only if QF[i,i] le QF[j,j] and
   2*|QF[i,j]| le M[i,i] for all i le j.  This is weaker
   than the matrix being Minkowski reduced. }

   dim := Degree(Parent(QF));
   for i in [1..dim-1] do
      if QF[i+1,i+1] lt QF[i,i] then
         return false;
      end if;
      for j in [i+1..dim] do
         if QF[i,i] lt 2*AbsoluteValue(QF[i,j]) then
            return false;
         end if;
      end for;
   end for;
   return true;
end intrinsic;

intrinsic IsLessThan(M1::AlgMatElt, M2::AlgMatElt) -> BoolElt
   {Compare two matrices under the ordering obtained by defining
   as smaller the matrix with the smallest first diagonal element,
   or, in absolute value, the largest off-diagonal element, under
   an ordering of the entries. }

   dim := Degree(Parent(M1));
   for i in [1..dim] do
      if M1[i,i] lt M2[i,i] then
         return true;
      else
         if M1[i,i] gt M2[i,i] then
            return false;
         end if;
      end if;
   end for;
   for j in [1..dim-1] do
      for i in [1..dim-j] do
         if AbsoluteValue(M1[i,i+j]) gt AbsoluteValue(M2[i,i+j]) then
            return true;
         elif AbsoluteValue(M1[i,i+j]) lt AbsoluteValue(M2[i,i+j]) then
             return false;
         end if;
      end for;
   end for;
   // The entries of the two matrices are equal in absolute value.
   // The smaller matrix has positive entry in the highest position.
   for j in [1..dim-1] do
      for i in [1..dim-j] do
         if M1[i,i+j] gt M2[i,i+j] then
            return true;
         elif M1[i,i+j] lt M2[i,i+j] then
            return false;
         end if;
      end for;
   end for;
   return false; // They are equal.
```

```
end intrinsic;

intrinsic CompareForms(M1::AlgMatElt, M2::AlgMatElt) -> RngIntElt
   {Return 1 if M1 is less than M2, 0 if M1 and M2 are equal,
   and -1 if M2 is less than M1.}

   dim := Degree(Parent(M1));
   for i in [1..dim] do
      if M1[i,i] lt M2[i,i] then
         return 1;
      else
         if M1[i,i] gt M2[i,i] then
            return -1;
         end if;
      end if;
   end for;
   for j in [1..dim-1] do
      for i in [1..dim-j] do
         if AbsoluteValue(M1[i,i+j]) gt AbsoluteValue(M2[i,i+j]) then
            return 1;
         elif AbsoluteValue(M1[i,i+j]) lt AbsoluteValue(M2[i,i+j]) then
             return -1;
         end if;
      end for;
   end for;
   // The entries of the two matrices are equal in absolute value.
   // The smaller matrix has positive entry in the highest position.
   for j in [1..dim-1] do
      for i in [1..dim-j] do
         if M1[i,i+j] gt M2[i,i+j] then
            return 1;
         elif M1[i,i+j] lt M2[i,i+j] then
            return -1;
         end if;
      end for;
   end for;
   return 0; // They are equal.
end intrinsic;

/*
In the following function we make use of the following structure.

Let SignChange(QF,i) be the operation on QF defined by changing
the sign of the ith variable.  Since Prod_i SignChange(QF,i) = 1,
and there exists no other relations, the group generated by
SignChanges is isomorphic to {1,-1}^(n-1).

Moreover, it has a faithful representation on the sign values of
a set of n-1 indices {{i_1,j_1},{i_2,j_2},...,{i_n-1,j_n-1}} for
nondiagonal elements provided that for any subset of cardinality
r, the collective set of indices has cardinality at least r+1.

For a subset S of r distinct index sets, the number s such that
the collective set of indices has cardinality r + s is said to be
the number of components of S.
*/

function PositiveReduction(QF)
   // PRE: Takes a symmetric matrix over the Integers().
   // POST: Defines an ordering of the non-diagonal elements
   // and makes as many as possible positive, with priority
   // given to the elements of lowest position in the ordering.

   RMat := Parent(QF);
   I := Identity(RMat);
```

```
   dim := Degree(RMat);

   FF2 := FiniteField(2);
   VF2 := VectorSpace(FF2,dim);

   PrioritySet := { };
   BoundaryBasis := { VF2 | };
   count := 0;
   // Go through the indices in desired order:
   for j in [1..dim-1] do
      for k in [1..dim-j] do
         WF2 := sub< VF2 | BoundaryBasis, VF2.k - VF2.(k+j) >;
         if Rank(WF2) gt count and QF[k,k+j] ne 0 then
            PrioritySet := PrioritySet join {[k,k+j]};
            BoundaryBasis := BoundaryBasis join {VF2.k - VF2.(k+j)};
            count := count + 1;
         end if;
      end for;
   end for;
   SkewBasis := { VF2 | };
   for x in PrioritySet do
      if QF[x[1],x[2]] lt 0 then
         SkewBasis := SkewBasis join {VF2.x[1] + VF2.x[2] + VF2.1};
      else
         SkewBasis := SkewBasis join {VF2.x[1] + VF2.x[2]};
      end if;
   end for;
   WF2 := sub< VF2 | SkewBasis >;
   UF2 := VectorSpace(FiniteField(2),Dimension(WF2));
   HF2 := Hom(UF2,VF2);
   w := Kernel(Transpose(HF2![WF2.i : i in [1..Dimension(WF2)]])).1;
   // Now do the sign changes.
   P := Identity(Parent(QF));
   for i in [1..dim] do
      if w[i] eq One(FF2) then
         P[i,i] := -P[i,i];
      end if;
   end for;
   if P*QF*P eq QF then
      P := I;
   end if;
   return P*QF*P, P;
end function;

function PermutationReduction(QF)
   // PRE: QF is a quadratic form.
   // POST: Returns the minimal form under the permutations of
   // the basis elements.

   // Standard initializations.
   RMat := Parent(QF);
   dim := Degree(RMat);
   U := Identity(RMat);
   Q0 := QF;

   // Initialize basis permutation group variables.
   NormValues := {Integers() | }; // Putative successive minima.
   Indices := {[1..dim] | i : i in [1..dim]};  // Index set.
   IndexComps := {Parent({Indices | }) | };    // Index components.
   IndexPerms := Sym(Indices);    // Full permutation group on indices.
   CompPerms := {IndexPerms | };  // Generator set of component
                                  // perutation subgroup.

   // Find the sets of indices on which QF[i,i] is constant.
   for i in [1..dim] do
```

```
      if QF[i,i] in NormValues then
         for Y in IndexComps do
            s := Random(Y);
            if QF[i,i] eq QF[s,s] then
               Z := Y;
            end if;
         end for;
         Exclude(~IndexComps,Z);
         Include(~Z,i);
         Include(~IndexComps,Z);
      else
        Include(~IndexComps,{i});
        Include(~NormValues,QF[i,i]);
      end if;
   end for;

   // Calculate the permutation group stabilizing components.
   for Y in IndexComps do
      y := Random(Y);
      Z := Exclude(Y,y);
      Gens := {IndexPerms | (z,y) : z in Z};
      CompPerms := CompPerms join Gens;
   end for;
   G := PermutationGroup<Indices | CompPerms>;

   // Pick the smallest form.
   for g in G do
      P := Zero(RMat);
      for i in [1..dim] do
         P[i,i^g] := 1;
      end for;
      Q1 := P*QF*Transpose(P);
      if IsLessThan(Q1,Q0) then
         Q0 := Q1; U := P;
      end if;
   end for;
   return Q0, U;
end function;

intrinsic InnerProduct(M::AlgMatElt,x::ModTupRngElt,y::ModTupRngElt)  -> RngElt
   {Returns the InnerProduct(x*M,y) with respect to M.}
   return InnerProduct(x*M,y);
end intrinsic;

function NeighborReduction(QF)
// {Returns the smallest form from those in the neighborhood
//  of QF.  The resulting form is Minkowski reduced.}

   RMat := Parent(QF);
   I := Identity(RMat);
   M := BaseModule(RMat);
   dim := Degree(RMat);

   // Neighbor search variables.
   LocalNeighbors := [ Parent({ Parent([ Integers() | ]) | }) | ];
   for i in [1..dim] do
      NearZero := {-1,0,1};
      Zeros := [ 0 : j in [i+1..dim] ];
      if i gt 1 then
          FreeHood := { x : x in CartesianPower(NearZero,i-1) };
          S := { [x[k] : k in [1..i-1]] cat [1]
                        cat Zeros : x in FreeHood };
      else
          S := { [1] cat Zeros };
      end if;
```

```
      for s in S do
         x := M!s;
         if InnerProduct(QF,x,x) lt QF[i,i] then
            // Found smaller ith basis vector.
            B0 := I; B0[i] := x;
            Q0, U0 := NormEchelon(B0*QF*Transpose(B0));
            return Q0, U0*B0;
         else
            if InnerProduct(QF,x,x) gt QF[i,i] then
               // Neighbor larger, exclude.
               Exclude(~S,s);
            end if;
         end if;
      end for;
      // S now consists only of coordinates of neighbors which
      // are ith successive minima.
      Append(~LocalNeighbors,S);
   end for;

   NeighborSpace := car< LocalNeighbors[1] >;
   for i in [2..dim] do
      NeighborSpace := Flat(car< NeighborSpace, LocalNeighbors[i] >);
   end for;

   for C in NeighborSpace do
      B0 := I;
      for i in [1..dim] do
         B0[i] := M![ C[i][j] : j in [1..dim] ];
      end for;
      Q0 := B0*QF*Transpose(B0);
      if IsLessThan(Q0,QF) then
         U := B0;
         Q0, U0 := NormEchelon(Q0);
         U := U0*B0;
         Q0, U0 := PositiveReduction(Q0);
         U := U0*B0;
         return Q0, U;
      end if;
   end for;
   return QF, I;
end function;

intrinsic ClassicalAdjoint(M::AlgMatElt) -> AlgMatElt
   {Returns the matrix N such that M*N = N*M is the determinant
   times the identity matrix.}
   n := Degree(Parent(M));
   Adj := Zero(Parent(M));
   for i in [1..n] do
      for j in [1..n] do
         Adj[j,i] := (-1)^(i+j)*MinorDeterminant(M,i,j);
      end for;
   end for;
   return Adj;
end intrinsic;

intrinsic MinorDeterminant(M::AlgMatElt,i::RngIntElt,j::RngIntElt) -> RngElt
   {Returns the determinant of the matrix obtained by removing
   the ith row and jth column from M.}
   n := Degree(Parent(M));
   R := BaseRing(Parent(M));
   N := Zero(MatrixAlgebra(R,n-1));
   d := 0;
   for r in [1..n-1] do
      d +:= KroneckerDelta(i,r);
      e := 0;
```

```
      for s in [1..n-1] do
         e +:= KroneckerDelta(j,s);
         N[r,s] := M[r+d,s+e];
      end for;
   end for;
   return Determinant(N);
end intrinsic;

intrinsic KroneckerDelta(x::.,y::.) -> RngElt
   {Returns 1 if x and y are equal and 0 otherwise.}
   if x cmpeq y then
      return 1;
   else
      return 0;
   end if;
end intrinsic;
```