

# 1 Code

```
RequirePackage("chevie");
RequirePackage("vkcurve");

W:=ComplexReflectionGroup(8);
b:=Mvp("b");
c:=Mvp("c");
d:=Mvp("d");
H:=Hecke(W,[[1,b,c,d]]);
R:=Representations(H);
L:=Length(R);
S:=SchurElements(H);

R:=Representations(H);
L:=Length(R);
S:=SchurElements(H);

CharIrr := CharTable(W).irreducibles;
Hyp := HyperplaneOrbits(W);
Params := H.parameter;

F:=function(g) local res,i,l,R,L,S;
    R:=Representations(H);
    L:=Length(R);
    S:=SchurElements(H);
    if g=[] then res:=Sum(List([1..L],l->Length(R[l][1])/S[l]));
    else res:=Sum(List([1..L],l->Trace(Product(List(g,
        i->R[l][AbsInt(i)]^(SignInt(i)))))/S[l]));
    fi;
return res;
end;

m:=function(W,i,s)local char,s,classnos,o;
    char:=CharTable(W).irreducibles[i];
    s:=HyperplaneOrbits(W)[s];
    classnos:=Concatenation([1],s.classno);
    o:=s.e_s;
    return List([0..o-1],j->Sum([0..o-1],
        k->char[classnos[k+1]]*E(o)^(-k*j))/o);
end;

omegapi:=function(W,H,i) local d,j,s,t,C,r,o,M,res;
    d:=CharTable(W).irreducibles[i][1];
    res:=1;
    for j in HyperplaneOrbits(W) do
        s:=j.s;
        t:=H.parameter[s];
        C:=HyperplaneOrbits(W)[s];
```

```

o:=C.e_s;
M:=m(W,i,s);
res:=res*Product(List([1..o],
    k->((E(o)^(-k+1)*t[k])^(M[k]*C.N_s*C.e_s*d^-1))));
od;
return res;
end;

G:=function(g) local res,i,l,ginv,R,L,S;
R:=Representations(H);
L:=Length(R);
S:=SchurElements(H);
if g=[] then
res:=Sum(List([1..L],l->(omegapi(W,H,l)*Length(R[l][1])/S[l]));
else
ginv:=(-1)*Reversed(g);
res:=Sum(List([1..L],l->(omegapi(W,H,l)*Trace(Product(List(ginv,i->R[l]
[AbsInt(i)]^(SignInt(i)))))/S[l]));
fi;
return res;
end;

Bas8:=[ [], [ 1 ], [ 1, 1 ], [ 1, 1, 1 ], [ 2 ], [ 2, 1 ], [ 2, 1, 1 ],
[ 2, 1, 1, 1 ], [ 1, 2 ], [ 1, 2, 1 ], [ 1, 2, 1, 1 ], [ 1, 2, 1, 1, 1 ],
[ 1, 1, 2 ], [ 1, 1, 2, 1 ], [ 1, 1, 2, 1, 1 ], [ 1, 1, 2, 1, 1, 1 ],
[ 1, 1, 1, 2 ], [ 1, 1, 1, 2, 1 ], [ 1, 1, 1, 2, 1, 1 ],
[ 1, 1, 1, 2, 1, 1, 1 ], [ 2, 2 ], [ 1, 2, 2 ], [ 1, 1, 2, 2 ],
[ 1, 1, 1, 2, 2 ], [ 1, 2, 1, 2, 1, 2 ], [ 1, 1, 2, 1, 2, 1, 2 ], [ 1, 1, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 1, 2, 1, 2, 1, 2 ], [ 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 2, 1, 2, 1, 2 ], [ 2, 1, 1, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 1, 2, 1, 2, 1, 2 ], [ 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ], [ 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ], [ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ], [ 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ], [ 2, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 2, 1, 2, 1, 2, 1, 2 ], [ 1, 1, 2, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2 ], [ 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2 ],

```

```

[ 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 1, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 1, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 1, 2, 1, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ],
[ 1, 1, 1, 2, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2, 1, 2 ];

```

```

PrintTo("dataR",R);
PrintTo("dataS",S);
PrintTo("basis",Bas8);
PrintTo("dataCharIrr",CharIrr);
PrintTo("dataHyp", List([1..Length(Hyp)],l->[Hyp[l].s, Hyp[l].e_s,
      Hyp[l].classno, Hyp[l].N_s, Hyp[l].det_s]));
PrintTo("dataParams",Params);

```

## 2 Short description

First we import the necessary GAP3 packages, define the variables and define the Hecke algebra we need. In this example we define the Hecke algebra of  $G_8$  at variables  $b$ ,  $c$  and  $d$ . Note that originally the algebra has four variables. But

by dividing each relation satisfied by the generators by one of the variables, we can reduce the number of variables by one. This makes the calculations less computationally intensive. This corresponds to the relations  $(s-1)(s-b)(s-c)(s-d) = 0$  and  $(t-1)(t-b)(t-c)(t-d) = 0$  of the algebra, where  $s$  and  $t$  are the generators. Moreover, we extract the data necessary for the computation.

```
RequirePackage("chevie");
RequirePackage("vkcurve");

W:=ComplexReflectionGroup(8);
b:=Mvp("b");
c:=Mvp("c");
d:=Mvp("d");
H:=Hecke(W,[[1,b,c,d]]);
R:=Representations(H);
L:=Length(R);
S:=SchurElements(H);

R:=Representations(H);
L:=Length(R);
S:=SchurElements(H);

CharIrr := CharTable(W).irreducibles;
Hyp := HyperplaneOrbits(W);
Params := H.parameter;
```

The function `F` corresponds to the first condition (trace). Since the computations in `GAP3` take a long time to be completed, this function has been reimplemented in `Mathematica`.

```
F:=function(g) local res,i,l,R,L,S;
  R:=Representations(H);;
  L:=Length(R);;
  S:=SchurElements(H);;
  if g=[] then res:=Sum(List([1..L],l->Length(R[l][1])/S[l]));
  else res:=Sum(List([1..L],l->Trace(Product(List(g,
    i->R[l][AbsInt(i)]^(SignInt(i)))))/S[l]));
  fi;
return res;
end;
```

The next functions are helper functions for the computations related to the third condition.

```
m:=function(W,i,s)local char,s,classnos,o;
  char:=CharTable(W).irreducibles[i];
  s:=HyperplaneOrbits(W)[s];
  classnos:=Concatenation([1,s.classno]);
  o:=s.e_s;
  return List([0..o-1],j->Sum([0..o-1],
    k->char[classnos[k+1]]*E(o)^(-k*j))/o);
end;
```





```
PrintTo("dataR",R);
PrintTo("dataS",S);
PrintTo("basis",Bas8);
PrintTo("dataCharIrr",CharIrr);
PrintTo("dataHyp", List([1..Length(Hyp)],l->[Hyp[l].s, Hyp[l].e_s,
      Hyp[l].classno, Hyp[l].N_s, Hyp[l].det_s]));
PrintTo("dataParams",Params);
```