MATHEMATICS OF COMPUTATION
VOLUME 64, NUMBER 211
JULY 1995, PAGES S35–S37

Supplement to
CONJUGACY CLASSES OF Γ(2) AND SPECTRAL RIGIDITY

RALPH PHILLIPS

```
program rigidity (input, output);
   {program for 1000 or less trace numbers}

type dataset = array [1..30000] of integer;

const  pi = 3.14159265359;
       d = 50;          {computing for d characters}

var  i,j,e,n,max,p,q,r1,r2,s,t,u: integer;  {matrix: mx = (a b, c d)}
     e2,a2,b2,c2,a3,b3,s1,u1,v1,w1,x1: real;
     a,b,c,f,a1,b1,c1,o1: dataset;    {mx[i] obtained by o1[i] conjugacy}
     m: array [0..1000] of integer;    {m counts matrices in list}
     x,y,z: array [0..1000,1..d] of real; {z counts conj. classes in n-chain}
     k,l,r: array [1..d] of integer;  {original character parameters}
     g,h: array [1..d] of real;    {modified character parameters}
  {definition: scod (sign change in off diagonal) of mx is (a -b, -c d)}

procedure info;
  var i: integer;
  begin
    writeln('give the maximum trace number');
    readln(max);
    for i:= 1 to d do
      begin
     writeln('for i = ', i:1,'  give character parameters: k, l, r');
        readln(k[i],l[i],r[i]);
      end;
    for i:= 0 to 1000 do
      for j:= 1 to d do
       begin
         x[i,j]:= 0;
         y[i,j]:= 0;
         z[i,j]:= 0;
       end;
    for i:= 0 to 1000 do
       m[i]:= 0;
    for i:= 1 to d do
     begin
      g[i]:= 0; h[i]:= 0;
     end;
  end;

procedure list(n: integer); {computes all matrices with 0 < a < tr}
  var e,i,j,x,y,tr,r: integer;    {having positive elements 0 < b <= c}
  begin
    r:= 0;  tr:= 4*n + 2;
    for j:= 0 to n do
     begin
      x:= 2*j + 1;          {possible value for a, 0 < a <= 2n+1}
      e:= (x*(tr - x) - 1) div 4;
      for i:= 1 to round(sqrt(e)) do
       begin
         y:= e mod i;
         if (y = 0) then  {if i divides e then compute matrix elements}
           begin
            r:= r + 1;
            a[r]:= x;
            b[r]:= 2*i;
            c[r]:= 2*e div i;
            if (x <> tr - x) then
```

```
          begin      {computes matrices for 2n+1 < a < 4n+2}
           r:= r + 1;
           a[r]:= tr - x;
           b[r]:= 2*i;
           c[r]:= 2*e div i;
          end;
        end;
      end;
     end;
    m[n]:= r;
 end;

procedure chara;  {character = exp(2*pi*i*(g*p + h*q))}
 begin
  if o1[u] = 0 then p:= p + 1;
  if o1[u] = 1 then q:= q + 1;
  if o1[u] = 2 then p:= p - 1;
  if o1[u] = 3 then q:= q - 1;
end;

procedure link(i,n: integer);  {computes next conjugated matrix}
   var x,tr: integer;
 begin
     tr:= 4*n + 2;
     if e = o1[u] then  {can not follow a conjugation with its inverse}
        e:= (e + 1) mod 4;
     if e = 0 then    {conjugates with (1 2, 0 1)}
        begin
           a1[u+1]:= a1[u] + 2*c1[u];
           b1[u+1]:= b1[u] + 2*tr - 4*a1[u] - 4*c1[u];
           c1[u+1]:= c1[u];
           o1[u+1]:= 2;
           u:= u + 1;
        end;
     if e = 1 then    {conjugates with (1 0, -2 1)}
        begin
           a1[u+1]:= a1[u] + 2*b1[u];
           b1[u+1]:= b1[u];
           c1[u+1]:= c1[u] + 2*tr - 4*a1[u] - 4*b1[u];
           o1[u+1]:= 3;
           u:= u + 1;
        end;
     if e = 2 then    {conjugates with (1 -2, 0 1)}
        begin
           a1[u+1]:= a1[u] - 2*c1[u];
           b1[u+1]:= b1[u] - 2*tr + 4*a1[u] - 4*c1[u];
           c1[u+1]:= c1[u];
           o1[u+1]:= 0;
           u:= u + 1;
        end;
     if e = 3 then   {conjugates with (1 0, 2 1)}
        begin
           a1[u+1]:= a1[u] - 2*b1[u];
           b1[u+1]:= b1[u];
           c1[u+1]:= c1[u] - 2*tr + 4*a1[u] - 4*b1[u];
           o1[u+1]:= 1;
           u:= u + 1;
        end;
     if (0 < a1[u]) and (a1[u] < tr) then {if 0 < a < tr}
       begin
           x:= 0;
           repeat   {checks whether mx[u] is in list matrices}
             if x < m[n] then
               x:= x + 1
           until (a[x] = a1[u]) and ((b[x] = abs(b1[u]))
                      or (b[x] = abs(c1[u])));
           f[x]:= 1;  {eliminates mx[u] as a starting list matrix}
           if (a1[u] <> a[i]) or (b1[u] <> b[i]) then
             begin  {if mx[u] <> starting matrix then it belongs to chain}
               chara;
               e:= 0;
        if (a1[u] = a[i]) and ((b1[u] = - b[i]) or (abs(b1[u]) = c[i])) then
               t:= t + 1; {chain contains a transpose or scod of mx[i]}
             end
           else
             begin       {chain ends when mx1[u] = mx[i]}
               e:= 0;
               s:= 1;
               chara;
             end;
         end
       else
         if ((0 < a1[u] + c1[u]) and (a1[u] + c1[u] < tr))
          or ((0 < a1[u] - c1[u]) and (a1[u] - c1[u] < tr)) then
            begin   {condition for mx[u] to belong to chain}
              e:= 0;
              chara;
            end
         else    {mx[u] does not belong to the chain}
           begin
             e:= (e + 1) mod 4;
             u:= u - 1;
           end;
end;

procedure chain(n: integer);
 var  j: integer;
     e1: real;
 begin
   i:= 1;
   while (i <= m[n]) do
     begin
       p:= 0;  q:= 0;    {initializes (p,q) character parameters}
       f[i]:= 1;  {eliminates mx[i] as a future starting matrix}
       if  u > 20000 then  {memory saver}
         begin    {prevents u from getting too large}
           for j:= 1 to 30000 do   {reinitializes the mx1[j]'s}
            begin
              a1[j]:= 0; b1[j]:= 0; c1[j]:= 0; o1[j]:= 0;
            end;
           u:= 1;
         end;
       a1[u]:= a[i];  b1[u]:= b[i];  c1[u]:= c[i];
       e:= 0; s:= 0; t:= 0;  {starts a new chain}
       if (b[i] = c[i]) then t:= 1;
       repeat
         link(i,n);  {tries to find next link in chain}
       until (s = 1);
       for j:= 1 to d do
         begin
```

```
        e1:= cos(g[j]*p + h[j]*q) + cos(g[j]*q + h[j]*p);
        if (t = 0) then z[n,j]:= z[n,j] + 2*e1;
          {transpose and scod give separate chains}
        if (t = 1) then z[n,j]:= z[n,j] + e1;
          {transpose or scod do not give separate chains}
        if (t >= 2) then z[n,j]:= z[n,j] + e1/2;
          {transpose and scod both lie in the same chain}
      end;
     i:= 1;
     repeat  {finds an unused initial matrix in list for a new chain}
        i:= i + 1;
     until (f[i] = 0);
   end;
 end;

begin
  info;
  for i:= 1 to d do
    begin     {computes modified character parameters}
      g[i]:= 2*pi*k[i]/r[i];  h[i]:= 2*pi*l[i]/r[i];
    end;
  for n:= 1 to max  do {computes conjugacy classes for nth trace}
    begin
      for i:= 1 to 30000 do  {initializes the matrix arrays}
        begin
         a[i]:= 0; b[i]:= 0; c[i]:= 0; f[i]:= 0;
         a1[i]:= 0; b1[i]:= 0; c1[i]:= 0; o1[i]:= 4;
        end;
      list(n);
      u:= 1;
      chain(n);
    end;
  for i:= 1 to d do {subtracts off nonprimitive conjugacy classes}
    begin
      for j:= 1 to 15 do  {2nd power}
        z[(4*j*j + 4*j),i]:= z[(4*j*j + 4*j),i] - z[j,i];
      for j:= 1 to 3 do  {3rd power}
       begin
         s:= j*j*j;
     z[(16*s + 24*j*j + 9*j),i]:= z[(16*s + 24*j*j + 9*j),i] - z[j,i];
       end;
      for j:= 1 to 1 do  {4th power}
        begin
          s:= j*j*j;  t:= j*j*j*j;
   z[(64*t+128*s+80*j*j+16*j),i]:= z[(64*t+128*s+80*j*j+16*j),i] - z[j,i];
        end;
    end;
  writeln(chr(7), chr(7), chr(7));
  a2:= 150;
  for i:= 1 to d do
   begin
    for j:= 1 to max  do {sum of squares of 'multiplicities'}
     begin
      y[j,i]:= y[j-1,i] + z[j,i]; {an incidental calculation}
      x[j,i]:= x[j-1,i] + sqr(ln(4*j+2))*sqr(z[j,i]);
     end;
    writeln('max = ',max:4,'   k,l,r = ', k[i]:2,'  ',l[i]:2,'  ',r[i]:2);
    writeln('slopes for sets of 150 trace numbers incremented by 15');
    u1:= 0;    v1:= 0;     s:= max - 150;
    while (s <= max) do  {best linear fit calculation}
      begin
        b2:= 0;  c2:= 0; a3:= 0; b3:= 0;
        for j:= s-149  to  s  do
         begin
          b2:= b2 + ln(4*j+2);
          c2:= c2 + sqr(ln(4*j+2));
          a3:= a3 + ln(x[j,i]);
          b3:= b3 + ln(x[j,i])*ln(4*j+2);
         end;
        e2:= sqr(b2) - a2*c2;
        s1:= (b2*a3 - a2*b3)/e2; {slope of best fit for set}
        u1:= u1 + s1;
        v1:= v1 + sqr(s1);
        write( s1:5:3,'   ');
        s:= s + 15;
       end;
     writeln;
     w1:= u1/11;   {average slope}
     x1:= sqrt(v1/11 - sqr(w1));  {sqrt of variance}
     writeln('average slope = ', w1:5:3, '   probable error = ', x1:5:3);
     writeln('total number of conjugacy classes = ', y[max,i]:10:0);
     writeln;
   end;
end.
```