The Matlab program consists of 6 .m files in the attachments. To run the codes, quaternion toolbox is needed, see S. J. Sangwine and N. Le Bihan, "Quaternion Toolbox for Matlab®," [Online], 2005, software library available at: http://qtfm.sourceforge.net/.

For example, to compute sample points for N=12, s=1, just input samplepoint(12,1) as follows.

```
>> vsamples = samplepoint(12,1)

for k = 1:12
    vsamples(k)
end
vsamples =
     1x12 quaternion array
ans =
      0 + 1.029 * I + 0 * J + 0 * K
ans =
      0 + 1.226 * I + 0 * J + 0 * K
ans =
      0 + 1.514 * I + 0 * J + 0 * K
ans =
      0 + 1.8 * I + 0 * J + 0 * K
ans =
      0 + 2.034 * I + 0 * J + 0 * K
ans =
      0 + 2.184 * I + 0 * J + 0 * K
ans =
      0 - 0.02905 * I + 1.028 * J + 0 * K
ans =
      0 - 0.2464 * I + 1.201 * J + 0 * K
ans =
      0 - 0.5937 * I + 1.392 * J + 0 * K
ans =
      0 - 0.9513 * I + 1.528 * J + 0 * K
ans =
      0 - 1.242 * I + 1.611 * J + 0 * K
ans =
      0 - 1.427 * I + 1.653 * J + 0 * K
>>
```

```
function [isolated_real,isolated_nreal,spherical] = q_roots(qpolycoe)
% q_roots computing all zeros of quaternion simple polynomial.
% Implementation of
% D. Janovsk?a and G. Opfer, “A note on the computation of all zeros of simple↙
quaternionic polynomials,”
% SIAM J. Numer. Anal., vol. 48, no. 1, pp. 244 - 256, 2010.

% Input1:  qpolycoe--coefficient vector of quaternion simple polynomial
% (from high order to low order).
%
% Output1: real isolated zeros.
% Output2: non-real isolated zeros.
% Output3: spherical zeros.


if ~isvector(qpolycoe)
     error('Error. Input must be a vector.')
end

epsl = 10^(-8); % use to chop real numbers that are close to zero by the exact↙
integer 0.


n = length(qpolycoe)-1;  % The order of quaternion polynomial
m = 2*n;  % The order of  its companion polynomial

%  reverse the coefficient vector such that they are rearranged  from low order to↙
high order.
qpolycoer = reshape(qpolycoe,1,n+1);
qpolycoerev = fliplr(qpolycoer);
qpolycoecev = reshape(qpolycoerev,n+1,1);


% computing the coefficients of its companion polynomial
bM1 = repmat(conj(qpolycoerev),n+1,1);
bM2 = repmat(qpolycoecev,1,n+1);
bM = bM1.*bM2;
%bM =scalar(bM);
bM =rot90(bM);
```

```
b = zerosq(1,m+1);



for k = 1:m+1
    b(k) = sum(diag(bM,k-n-1));
end

%brev = scalar(fliplr(b));
b = scalar(b);


% computing the zeros of its companion polynomial
syms x l

F(x) = sum(b.*subs(x^l,l,0:m));
S1 = vpasolve(F(x)==0,x);        % vpasolve is more accurate than roots, but is
still not accurate enough.

d = double(S1);




d =  reshape(d,1,m);
d(abs(d)<epsl) = 0;



% pick up the real-valued zeros.

zr = d(abs(imag(d))<epsl);
zr = real(zr);
zr = sort(zr);
ind1 = ones(1,length(zr));

for k = 2:length(zr)
    if abs(zr(k)-zr(k-1))<epsl
        ind1(k)=0;
    end
end
```

```
zr(ind1==0) = [];

% pick up non-real zeros.

newd = d(imag(d)>eps1);
newd = sort(newd,'ComparisonMethod','abs');
ind2 = ones(1,length(newd));
%newd = unique(newd);

for k = 2:length(newd)
    if abs(newd(k)-newd(k-1))<eps1
        ind2(k)=0;
    end
end

newd(ind2==0) = [];


zi = setdiff(newd, zr);




n1 = length(zi);

zio = zerosq(1,n+1);   % save isolated non-real zeros
zis = zerosq(1,n+1);   % save spherical zeros


% The key step to find isolated non-real spherical zeros, more detail: see
% the reference  D. Janovsk?a and G. Opfer 2010.

for k = 1:n1
    u = real(zi(k))+ 1i*sqrt(abs(zi(k))^2 - real(zi(k))^2);
    alpha = imag(u.^(0:n))./sqrt(abs(zi(k))^2 - real(zi(k))^2);
    beta = zi(k).^(0:n) - alpha*zi(k);
    qalpha = quaternion(real(alpha),imag(alpha),zeros(1,n+1),zeros(1,n+1));
    qbeta = quaternion(real(beta),imag(beta),zeros(1,n+1),zeros(1,n+1));
    A = qalpha*qpolycoecev;
    B = qbeta*qpolycoecev;
    v = conj(A).*B;
```

```
    if abs(v)<epsl
        zis(k) =  quaternion(real(zi(k)),imag(zi(k)),0,0);
    else
        x0 = real(zi(k));
        x1 = abs(imag(zi(k)))*scalar(v*qi)/abs(vector(v));
        x2 = abs(imag(zi(k)))*scalar(v*qj)/abs(vector(v));
        x3 = abs(imag(zi(k)))*scalar(v*qk)/abs(vector(v));
        x0(abs(x0)<epsl) = 0;
        x1(abs(x1)<epsl) = 0;
        x2(abs(x2)<epsl) = 0;
        x3(abs(x3)<epsl) = 0;
        zio(k) =  quaternion(x0,x1,x2,x3);
    end
end

zio = zio(abs(zio)>epsl);
zis = zis(abs(zis)>epsl);

% return the different types of zeros.

isolated_real = zr;
isolated_nreal = zio;
spherical = zis;




end
```

```
function coeq = CoeQPoly(q,N,s)
% CoeQPoly Computes the coefficient vector (from low to high) of \langle \vphi(q,s)
,\vphi(\lambda,s)\rangle
%  - ix(k + 1) + jx(k) - ix(k - 1) = x(k)lambda

% Input-  N: k from 1 to N
%         q: lambda = q
%         s: initial value of x(1)
epsl = 10^(-9); % use to chop real numbers that are close to zero by the exact
integer 0.

coematrix = phicoe(N,s);

vphiq = vphi(q,N,s);

MDvPhi = repmat(vphiq,1,N);

Mq =  conj(MDvPhi).*coematrix;

coeq = sum(Mq);
x0 = scalar(coeq);
x1 = x(coeq);
x2 = y(coeq);
x3 = z(coeq);

 x0(abs(x0)<epsl) = 0;
 x1(abs(x1)<epsl) = 0;
 x2(abs(x2)<epsl) = 0;
 x3(abs(x3)<epsl) = 0;

coeq = quaternion(x0,x1,x2,x3);
end
```

```
function samples = samplepoint(N,s)
% samplepoint computes sample points associated with
%  - ix(k + 1) + jx(k) -ix(k - 1) = x(k)lambda



 [~,coePNs] = phicoe(N,s); % compute the coefficients of p_N(lambda,s) by phicoe.

 coePNs = fliplr(coePNs);


 [S1,S2,S3] = q_roots(coePNs); % compute the zeros of p_N(lambda,s) by q_roots.


 n1 = length(S3);
 n2 = N - length(S1) - length(S2) - length(S3) + 1;


 newS = zerosq(n1,n2);

 % compute sample points by Method 2 described in the paper.

for k1 = 1:n1
    q = S3(k1);
    newS(k1,1) = q;
    coeq = CoeQPoly(q,N,s);
    coeq = fliplr(coeq);
    [~,W1,~] = q_roots(coeq);
    W = W1(abs(scalar(W1)-scalar(q)) + abs(abs(vector(W1)) - abs(vector(q)))<10^
(-4));
    if isempty(W)
        continue
    end
    W = q_sort(W);
    W2 = W;
    newS(k1,2) = W2(1);
    n3 = length(W2);
    Ind = ones(1,n3);
    for k2 = 2:n3
```

```
        T = zerosq(1,k2-1);
        for k3 = 1:k2-1
            T(k3) = innerprod_vphi(W(k3),W2(k2),N,s)*Ind(k3);
        end
        normT = sum(abs(T));
        if normT<10^(-4)
            newS(k1,1+k2) = W2(k2);
        else
            Ind(k2) = 0;
        end
    end

end

newS = reshape(newS,1,n1*n2);

newS3 = newS(abs(newS)>10^(-6));


samples = [S1,S2,newS3];


end
```

```
function innerprod = innerprod_vphi(a1,a2,N,s)
%    innerprod_vphi computes inner product of vphi(a1,N,s) and vphi(a2,N,s)
%    related to - ix(k + 1) + jx(k) - ix(k - 1) = x(k)lambda



innerprod = reshape(conj(vphi(a1,N,s)),1,N)*reshape(vphi(a2,N,s),N,1);





end
```

```
function vphiq = vphi(q,N,s)
% vphi computes the column vector (x(1),x(2),...,x(N))^T at lambda=q
%  - ix(k + 1) + jx(k) - ix(k - 1) = x(k)lambda

% Input-  N: k from 1 to N
%         q: lambda = q
%         s: initial value of x(1)


switch nargin
    case 2
       s = 1;
end

vq = q.^(0:N-1);

vq = reshape(vq,N,1);


coematrix = phicoe(N,s);

vphiq = coematrix*vq;









end
```

```
function [coematrix,coePNs] = phicoe(N,s)
% phicoe computes the coefficients of the solution for - ix(k + 1) + jx(k) - ix(k -↙
1) = x(k)lambda
% return the coefficient matrix for x(1) to x(N) andthe  coefficient of
% P_N(lambda,s).


% Input-  N: k from 1 to N
%         s: initial value of x(1)

% Output-  coematrix: N by N coefficient matrix for x(1) to x(N)
%          coePNs: coefficient of PNs  % from low to high.

switch nargin
    case 1
       s = 1;
end

CoePhi = zerosq(N+1);
CoePhi(2,2) = s;

for k = 3:N+2
    for n = 2:N+2
        if k>n
            CoePhi(k,n) = quaternion(0,1,0,0)*CoePhi(k-1,n-1) - quaternion(0,0,0,1)↙
*CoePhi(k-1,n) - CoePhi(k-2,n);
        elseif k==n
            CoePhi(k,n) = quaternion(0,1,0,0)*CoePhi(k-1,n-1);
        end
    end
end

coematrix = CoePhi(2:N+1,2:N+1);

coePNs = CoePhi(N+2,2:N+2); % from low to high.

end
```