```
> restart;
> with(LinearAlgebra):
  with(plots):
```

# 1D Model Problem

Construct the typical tridiagonal Matrix for the 1D Poisson equation with zero right hand side vector b:

```
> n:=7;
  A := Matrix( [ [ seq(-1, i=1..n-1) ],
                 [ seq( 2, i=1..n) ],
                 [ seq(-1, i=1..n-1) ] ], shape=band[1,1],
  scan=band[1,2]);
```

$$n := 7$$

$$A := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

```
> b := < seq(0, i=1..n) >;
```

$$b := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

```
> evalf(Eigenvalues(A));
```

$$\begin{bmatrix} 2. \\ 3.414213562 \\ 0.585786438 \\ 0.152240935 \\ 3.847759065 \\ 1.234633135 \\ 2.765366865 \end{bmatrix}$$

```
>
```

# Conjugate Gradients

Standard CG-algorithm withou preconditioning:

```
> CG := proc(A::Matrix, b::Vector, xstart::Vector, it::posint)
      #option trace;
      local i, alpha, beta, x, r, resnorm, d:
      x[0] := xstart;
      r := b - A.x[0];
      d := r;
```

```
    for i from 1 to it do
       resnorm:= DotProduct(r,r);
       alpha := resnorm/BilinearForm(d,d,A);
       x[i] := x[i-1] + alpha*d;
       r := r - alpha*A.d;
       beta := DotProduct(r,r)/resnorm;
       d := r + beta*d;
    end do;
    return x;
 end proc;
```

CG := proc(A::Matrix, b::Vector, xstart::Vector, it::posint)
local i, α, β, x, r, resnorm, d;
    x[0] := xstart;
    r := b − (A . (x[0]));
    d := r;
    for i to it do
        resnorm := LinearAlgebra:-DotProduct(r, r);
        α := resnorm / LinearAlgebra:-BilinearForm(d, d, A);
        x[i] := x[i − 1] + α*d;
        r := r − ((α*A) . d);
        β := LinearAlgebra:-DotProduct(r, r) / resnorm;
        d := r + β*d
    end do;
    return x
end proc

```
> xstart := evalf(RandomVector(n));
  # try an eigenvector instead:
  # k := 5;
  # xstart := < seq( evalf(sin(k*Pi*j/(n+1))), j=1..n) >;
```

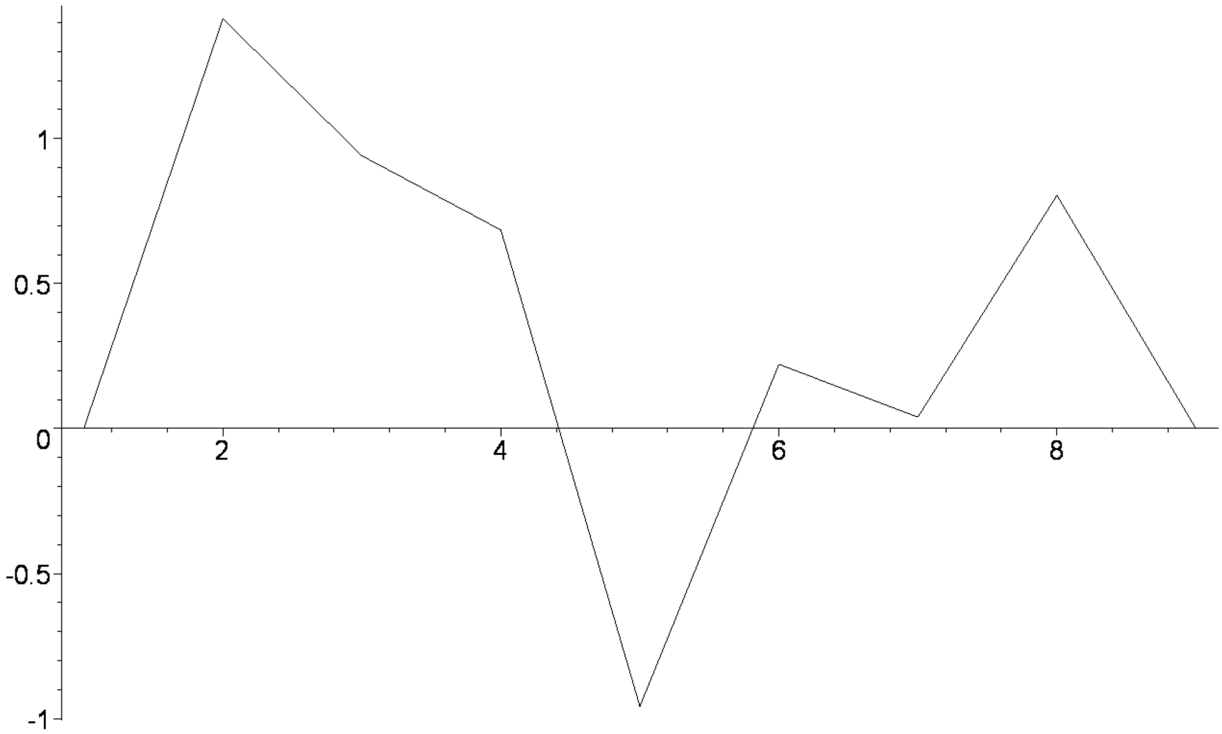
$$xstart := \begin{bmatrix} 27. \\ 8. \\ 69. \\ 99. \\ 29. \\ 44. \\ 92. \end{bmatrix}$$

```
> iter := 20;
  xs := CG(A,b,xstart,iter);
```

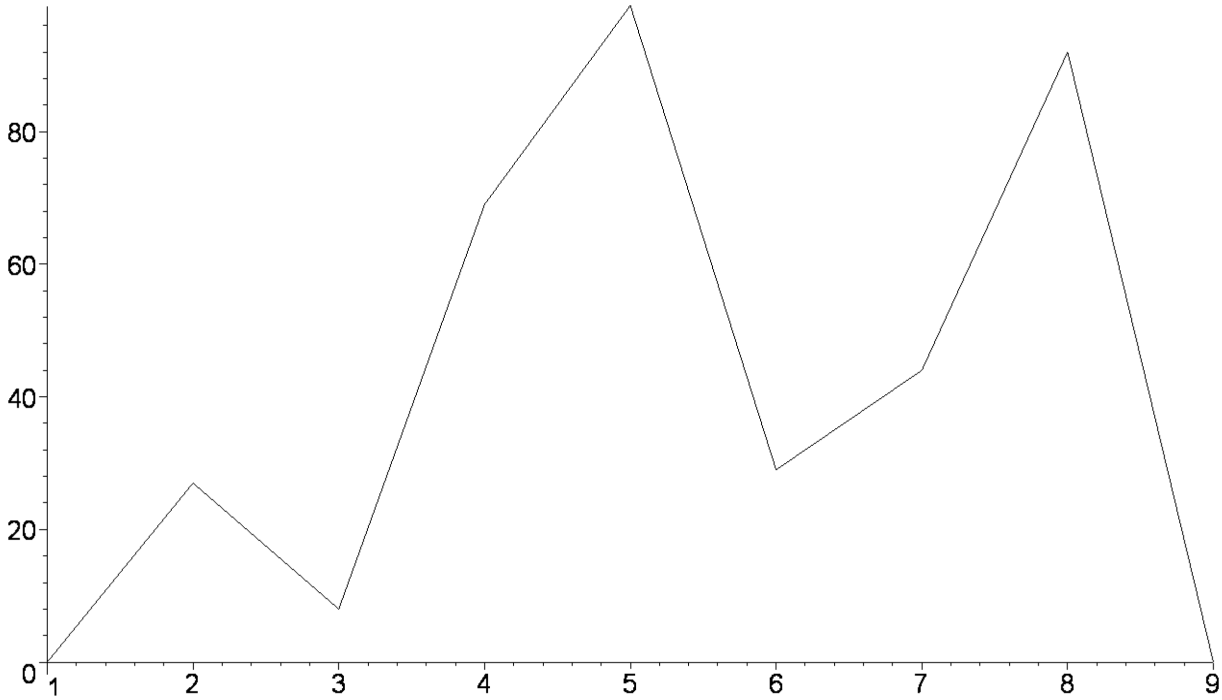
iter := 20

xs := x

```
> listplot([ 0, seq(xs[6][i], i=1..n), 0]);
```

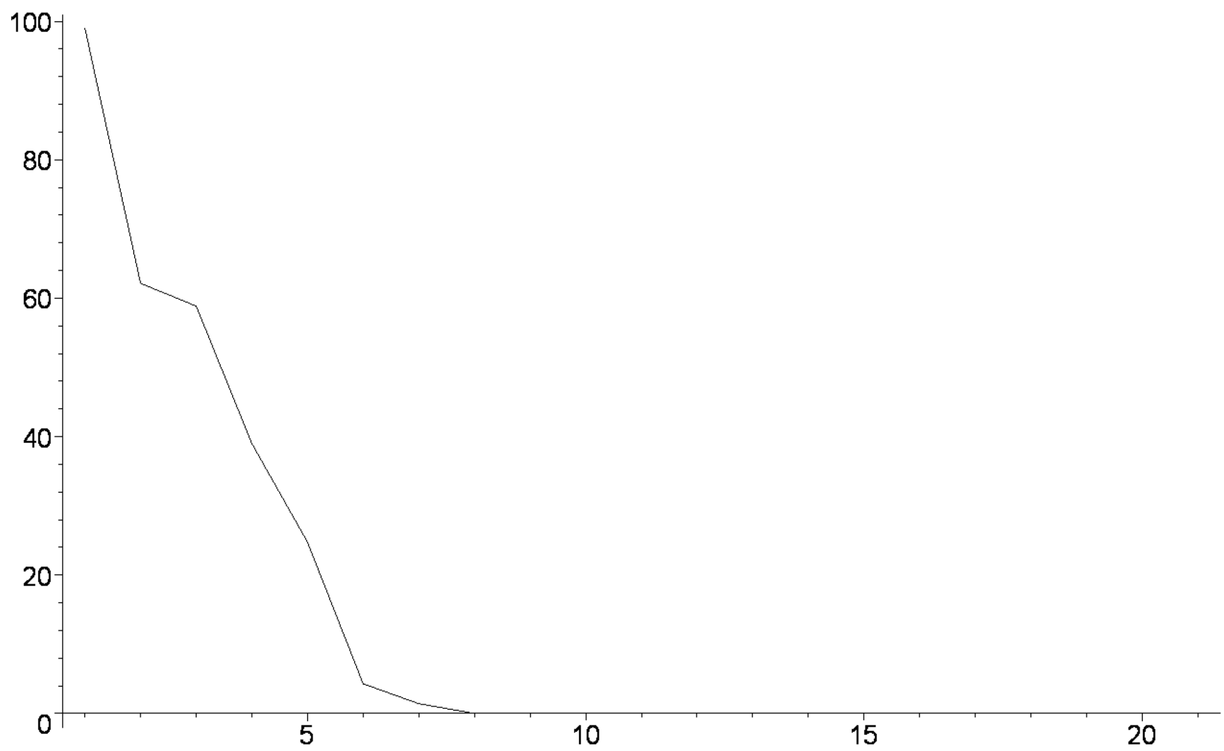



```
> solplot := [ seq( listplot([ 0, seq(xs[j][i], i=1..n), 0]),
  j=0..iter) ]:
  display(solplot, insequence=true);
```




plot the maximum errors of each iterative solution:

```
> listplot( [ seq( Norm(xs[j]), j=0..iter ) ]);
```




```
>
```

# Preconditioned Conjugate Gradient - Hierarchical Basis Preconditioning

```
>
```

## Hierarchical Basis Transformation

hierarch corresponds to the inverse of the matrix L of the lecture slides
(transforms coefficients for nodal basis into coefficients for hierarchical basis

```
> hierarch := proc( x::Vector, n::posint )
    # option trace;
    local l,i, left, right, xx;
    xx := evalf(x);
    l := 1;
    while l < n/2 do
      for i from l to n by 2*l do
        if i-l>0 then left := xx[i-l] else left:=0 end if;
        if i+l<=n then right := xx[i+l] else right:=0 end
  if;
        xx[i] := xx[i]-(left+right)/2;
      end do;
      l := l*2;
    end do;
    return xx;
  end proc:
```

dehierarch corresponds to matrix L on the lecture slides
(transforms coefficients for hierarchical basis into coefficients for nodal basis)

```
> dehierarch := proc( x::Vector, n::posint )
```

```
    #option trace;
    local l,i,left,right,xx;
    xx := evalf(x);
    l := (n+1)/4;
    while l >= 1 do
      for i from l to n by 2*l do
        if i-l>0 then left := xx[i-l] else left:=0 end if;
        if i+l<=n then right := xx[i+l] else right:=0 end
  if;
        xx[i] := xx[i]+(left+right)/2;
      end do;
      l := l/2;
    end do;
    return xx;
  end proc:
```

dehierarchT corresponds to the transpose of matrix L
(applied to a vector of nodal basis function, it will transform this into a vector of hierarchical basis functions)

```
> dehierarchT := proc( x::Vector, n::posint )
    #option trace;
    local l,i,left,right,xx;
    xx := evalf(x);
    l := 1;
    while l <= (n+1)/4 do
      for i from 2*l to n by 2*l do
        xx[i] := xx[i]+(xx[i-l]+xx[i+l])/2;
      end do;
      l := l*2;
    end do;
    return xx;
  end proc:
```

Example for hierarchization - use a sin-mode as example:

```
> k := 1;
  xstart := < seq( evalf(sin(k*Pi*j/(n+1))), j=1..n) >;
```

$$k := 1$$

$$xstart := \begin{bmatrix} 0.3826834325 \\ 0.7071067810 \\ 0.9238795325 \\ 1. \\ 0.9238795325 \\ 0.7071067810 \\ 0.3826834325 \end{bmatrix}$$

```
> hierarch(xstart,n);
```

$$\begin{bmatrix} 0.0291300420 \\ 0.2071067810 \\ 0.0703261420 \\ 1. \\ 0.0703261420 \\ 0.2071067810 \\ 0.0291300420 \end{bmatrix}$$

```
> dehierarch(%,n);
```

$$\begin{bmatrix} 0.3826834325 \\ 0.7071067810 \\ 0.9238795325 \\ 1. \\ 0.9238795325 \\ 0.7071067810 \\ 0.3826834325 \end{bmatrix}$$

## CG with Hierarchical-Basis Preconditioning

```
> PCG := proc(n::posint, A::Matrix, b::Vector,
  xstart::Vector, it::posint)
     #option trace;
     local i, alpha, beta, x, r, resnorm, d, Ad:
     x[0] := xstart;
     r := dehierarchT(b - A.x[0],n);
     d := dehierarch(r,n);
     for i from 1 to it do
        resnorm:= DotProduct(r,r);
        Ad := A.d;
        alpha := resnorm/DotProduct(d,Ad);
        x[i] := x[i-1] + alpha*d;
        r := r - alpha*dehierarchT(Ad,n);
        beta := DotProduct(r,r)/resnorm;
        d := dehierarch(r,n) + beta*d;
     end do;
     return x;
  end proc:
> n := 7;
  A := Matrix( [ [ seq(-1, i=1..n-1) ],
                 [ seq( 2, i=1..n) ],
                 [ seq(-1, i=1..n-1) ] ], shape=band[1,1],
  scan=band[1,2]);
  b := < seq(0, i=1..n) >;
  xstart := evalf( RandomVector(n) );
  # k := 3;
  # xstart := < seq( evalf(sin(k*Pi*j/(n+1))), j=1..n) >;
```

$$n := 7$$

$$A := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

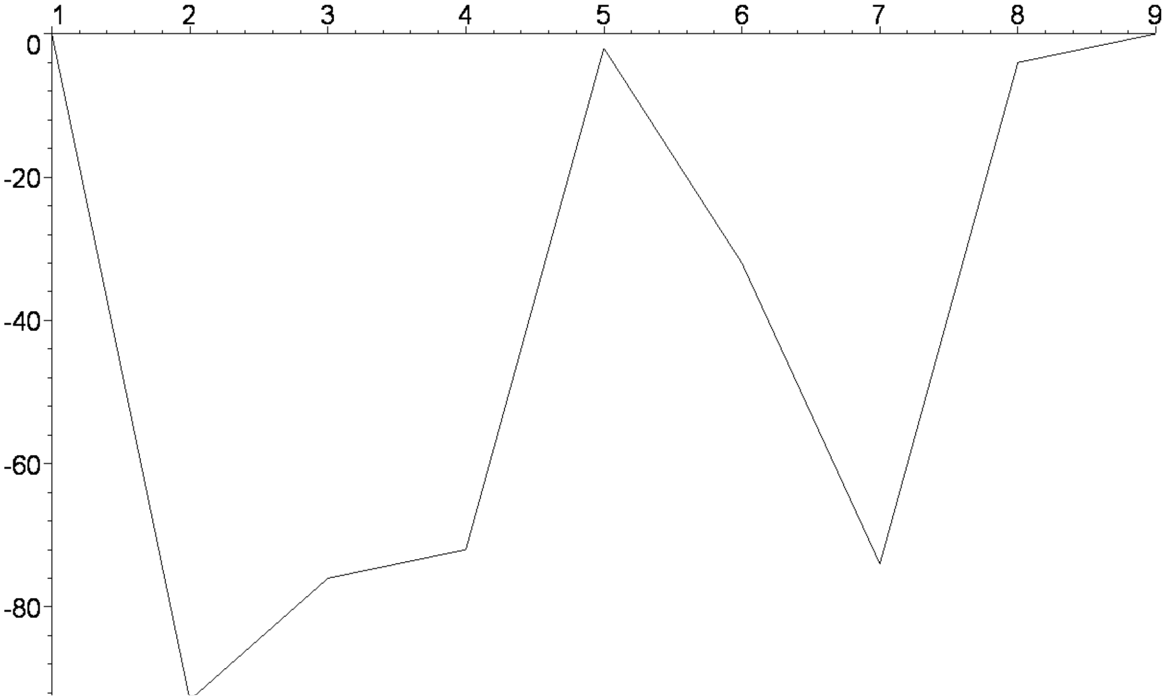
$$b := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$xstart := \begin{bmatrix} -93. \\ -76. \\ -72. \\ -2. \\ -32. \\ -74. \\ -4. \end{bmatrix}$$

```
> iter := 10;
  xs := PCG(n,A,b,xstart,iter);
  solplot := [ seq( listplot([ 0, seq(xs[j][i], i=1..n), 0]), j=0..iter) ]:
  display(solplot, insequence=true);
```

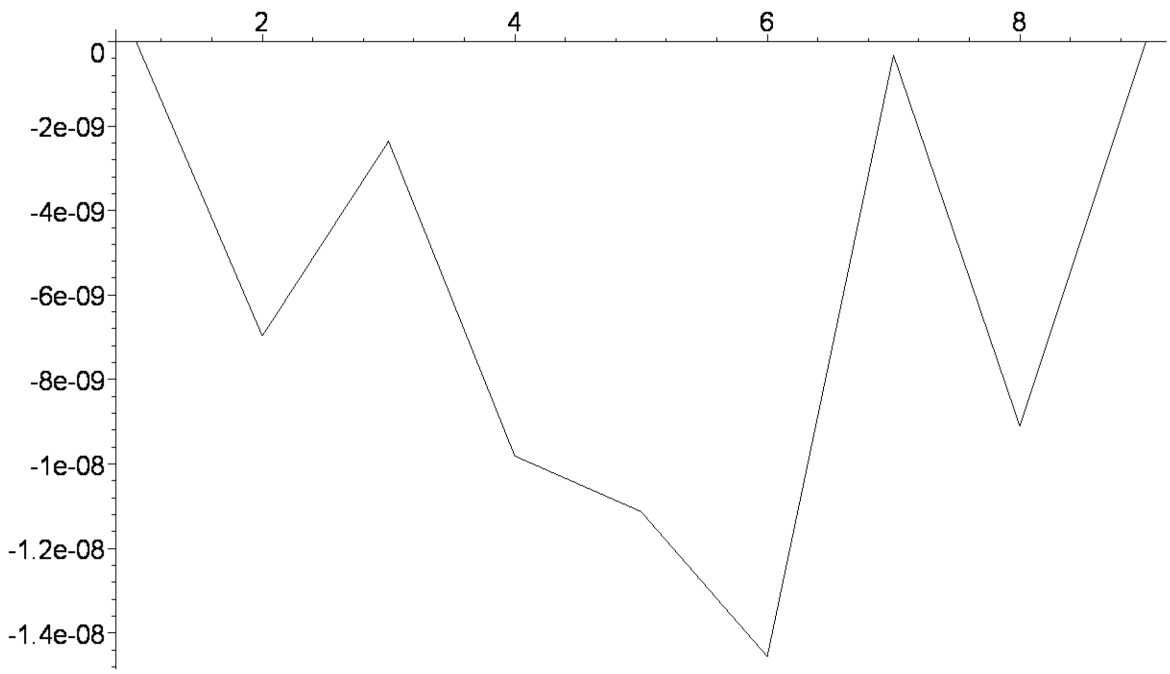
$$iter := 10$$

$$xs := x$$




Plot the solution after the first iteration -> basically =0 (hence, convergence in a single step).

```
> listplot([ 0, seq(xs[3][i], i=1..n), 0]);
```




# Explanation: Compute the Preconditioning Matrices

The matrices R (for the hierarchical transform) and L (for de-hierarchization) can be computed column-wise by applying the transforms to all unit vectors.
Note that in Maple notation <a,b,c> denotes a matrix, if a, b, and c are row vectors of the matrix.

```
> n := 7;
  A := Matrix( [ [ seq(-1, i=1..n-1) ],
                 [ seq( 2, i=1..n) ],
                 [ seq(-1, i=1..n-1) ] ], shape=band[1,1],
  scan=band[1,2]);
```

$$n := 7$$

$$A := \begin{bmatrix} 2 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & 2 \end{bmatrix}$$

```
> R := Transpose( < seq( Transpose( hierarch( < seq(
  `if`(i=j,1,0),j=1..n) >,n) ), i=1..n ) > );
```

$$R := \begin{bmatrix} 1. & -0.5000000000 & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & -0.5000000000 & 0. & 0. & 0. \\ 0. & -0.5000000000 & 1. & -0.5000000000 & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & -0.5000000000 & 1. & -0.5000000000 & 0. \\ 0. & 0. & 0. & -0.5000000000 & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & -0.5000000000 & 1. \end{bmatrix}$$

```
> L  := Transpose( < seq( Transpose( dehierarch( < seq(
```

```
`if`(i=j,1,0),j=1..n) >,n) ), i=1..n ) > );
```

$$L := \begin{bmatrix} 1. & 0.5000000000 & 0. & 0.2500000000 & 0. & 0. & 0. \\ 0. & 1. & 0. & 0.5000000000 & 0. & 0. & 0. \\ 0. & 0.5000000000 & 1. & 0.7500000000 & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0.7500000000 & 1. & 0.5000000000 & 0. \\ 0. & 0. & 0. & 0.5000000000 & 0. & 1. & 0. \\ 0. & 0. & 0. & 0.2500000000 & 0. & 0.5000000000 & 1. \end{bmatrix}$$

Note that R and L are inverse to each other (as hierarchical "forward" and "backward" transformations)

```
> R.L;
```

$$\begin{bmatrix} 1. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 1. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 1. \end{bmatrix}$$

Most important: L^T A L is a diagonal matrix (which explains the rapid CG convergence); note that the size of the diagonal levels depends on the hierarchical level of the corresponding basis functions!

```
> LAL := Transpose(L).Matrix(A).L;
```

$$LAL := \begin{bmatrix} 2. & 0. & 0. & 0. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. & 0. & 0. & 0. \\ 0. & 0. & 2. & 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0.500000000000000000 & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. & 2. & 0. & 0. \\ 0. & 0. & 0. & 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 0. & 0. & 0. & 2. \end{bmatrix}$$

```
> b := < seq(0, i=1..n) >;
  xstart := evalf( RandomVector(n) );
```

$$b := \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$xstart := \begin{bmatrix} -18. \\ 87. \\ 33. \\ -98. \\ -77. \\ 57. \\ 27. \end{bmatrix}$$

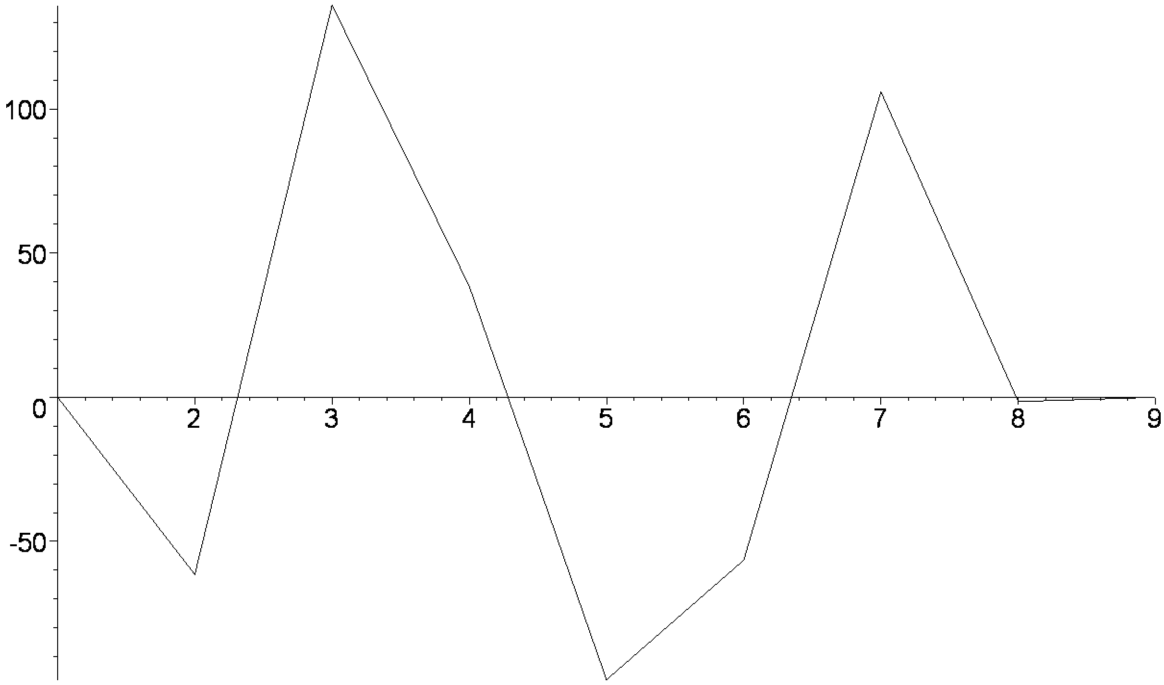
Apply regular CG algorithm to the transformed system of equations;

note that we transform the starting solution xstart into hierarchical-basis form:

```
> iter := 3;
  xs := CG(LAL,dehierarchT(b,n),hierarch(xstart,n),iter);
  solplot := [ seq( listplot([ 0, seq(xs[j][i], i=1..n),
  0]), j=0..iter) ]:
  display(solplot, insequence=true);
```

$$iter := 3$$

$$xs := x$$




```
>
```