restart;

SOR := proc(A::Matrix,b::Vector,w::numeric,x0::Vector,
tol::positive,n::nonnegint,v::name)
local AA, B, OK, N, I, J, X0, X1, TOL, NN, K, ERR, S, X, W;
with(LinearAlgebra);
N:=RowDimension(A);
AA:=A;
B:=b;
X0:=x0;
TOL:=tol;
NN:=n;
X1:=Vector(N);
W:=w;

STEP 1
K := 0;
OK := FALSE;
printf(` n x\n`);
printf(` - -\n`);
printf(`%3d [~,K);
for I from 1 to N do
printf(`% 12.8f`,X0[I]);
if I<>N then printf(`\n`) fi;
od;
printf(`\n`); K:=1;

STEP 2
while OK = FALSE and K <= NN do
err is used to test accuracy - it measures the infinity-norm
ERR := 0;
STEP 3
for I from 1 to N do
S := 0;
for J from 1 to I-1 do
S := S-A[I,J]*X1[J];
od;
for J from I+1 to N do
S := S-A[I,J]*X0[J];
od;
if abs(S-X0[I]) > ERR then
ERR := abs(S-X0[I]);
fi;
use X1 for X
X1[I] := S;
od;
printf(` %3d [~,K);
for I from 1 to N do
printf(`% 12.8f`,X1[I]);
if I<>N then printf(`\n`) fi;
od;
printf(`\n`);

STEP 4
if ERR <= TOL then
OK := TRUE;
fi;

process is complete
STEP 5
K := K+1;
STEP 6
> for I from 1 to N do
> X0[I] := X1[I];
> od;
> od;
> if OK = FALSE then
> printf(`Maximum Number of Iterations Exceeded.
`);
STEP 7
procedure completed unsuccessfully
> else
> printf(`The solution vector is
`);
> v:=evalm(X1);
> fi;
> end;

Warning, imaginary unit `I` used as a local variable in procedure SOR
SOR:= proc (A::Matrix, b::Vector, w::numeric, x0::Vector, tol::positive, n::nonnegint, v::name)
local AA, B, OK, N, I, J, X0, X1, TOL, NN, K, ERR, S, X, W;
with(LinearAlgebra);
N := RowDimension(A);
AA := A;
B := b;
X0 := x0;
TOL := tol;
NN := n;
X1 := Vector(N);
W := w;
K := 0;
OK := FALSE;
printf(` n  x 
`);
printf(` - - 
`);
printf(`%3d  
`, K);
for I to N do
printf(`% 12.8f`, X0[I]);
if I <> N then
printf(`
`)
end if;
end do;
printf(`
`
);
K := 1;
while $OK = FALSE$ and $K <= NN$ do

$ERR := 0;$

for $I$ to $N$ do

$S := 0;$

for $J$ to $I - 1$ do

$S := S - A[I, J] \times X1[J]$  
end do;

for $J$ from $I + 1$ to $N$ do

$S := S - A[I, J] \times X0[J]$  
end do;

$S := evalf((1 - W) \times X0[I] + W \times (S + B[I]) / A[I, I]);$

if $ERR \lt \text{abs}(S - X0[I])$ then

$ERR := \text{abs}(S - X0[I])$
end if;

$X1[I] := S$
end do;

printf(`\%3d `, $K);$  

for $I$ to $N$ do

printf(`\%12f`, $X1[I]);

if $I \lt > N$ then

printf(``$`)
end if
end do;

printf(`\`$`)

if $ERR \leq TOL$ then

$OK := TRUE$
end if;

$K := K + 1;$

for $I$ to $N$ do

$X0[I] := X1[I]$
end do
end do

if $OK = FALSE$ then

printf(`Maximum Number of Iterations Exceeded. `)
`
else

printf(`

The solution vector is
`);

v := evalm(X1)
end if
end proc

SOR_dir:=proc()

printf(`SOR returns an approximation to a solution of a vector equation.
`);

printf(`The arguments for SOR are:
`);

printf(`(1)the coefficient matrix (must be square)
`);

printf(`(2)the right hand side vector
`);

printf(`(3)the parameter omega
`);

printf(`(4)the initial approximation vector
`);

printf(`(5)tolerance
`);

printf(`(6)maximum number of iterations
`);

printf(`(7)variable for returning the approximate solution
`);

printf(`(8)If assigning the result to a variable, have the
`);

printf(`variable and the 7th argument the same.
`);

printf(`If v is the variable for returning the approximate
`);

printf(`solution and has already been given a value,
`);

printf(`the procedure should be preceded by the statement:
`);

printf(`v := `);

end;

SOR_dir := proc()
\texttt{printf(\texttt{(7)variable for returning the approximate solution})
\texttt{);}
\texttt{printf(\texttt{\texttt{\textbackslash 'If assigning the result to a variable, have the})
\texttt{\textbackslash '})
\texttt{printf(\texttt{\texttt{\textbackslash 'variable and the 7th argument the same.})
\texttt{\textbackslash '})
\texttt{printf(\texttt{\texttt{\textbackslash 'If v is the variable for returning the approximate solution})
\texttt{\textbackslash '})
\texttt{printf(\texttt{\texttt{\textbackslash 'and has already been given a value,})
\texttt{\textbackslash '})
\texttt{printf(\texttt{\texttt{\textbackslash 'the procedure should be preceded by the statement:})
\texttt{\textbackslash '})
\texttt{printf(\texttt{\texttt{\textbackslash 'v:=v'})
\texttt{end proc}