PSO Maple Code

```
restart;
with(RandomTools):

# ------------------------------
# Problem settings (user-configurable)
# ------------------------------
NP := 25:          # number of particles
Tmax := 60:       # maximum iterations
tol := 1e-8:       # significant change tolerance
noImproveLimit := 20:  # stop after this many iterations without
    improvement

# Search range for parameters
eps_min := 0.0:  eps_max := 10.0:
rho_min := 0.0:  rho_max := 1.0:

# ------------------------------
# Fitness function - replace with your actual problem
# ------------------------------
Fitness := proc(eps, rho)
    # Replace with your real fitness function
    return 0;
end proc:

# ------------------------------
# Helper functions
# ------------------------------
LogisticSequence := proc(z0, mu, N)
    local seq, z, i;
    seq := Array(0..N):
    z := z0:
    seq[0] := z:
    for i from 1 to N do
        z := mu*z*(1 - z):
        seq[i] := z:
    end do:
    return seq:
end proc:

SineWeightSequence := proc(x0, varrho, N)
    local seq, x, t, x_val;
    seq := Array(0..N):
    x := x0:
    seq[0] := x:
    for t from 1 to N do
        x_val := evalf((varrho/4)*sin(Pi*x)):
        if x_val <= 0 then x_val := 1e-6 fi:
        if x_val >= 1 then x_val := 1 - 1e-6 fi:
        x := x_val:
        seq[t] := x:
```

```
    end do:
    return seq:
end proc:

# ----------------------------
# Initialize PSO
# ----------------------------
InitializeSwarmPSO := proc(NP, eps_min, eps_max, rho_min, rho_max)
    local z0_eps, z0_rho, zseq_eps, zseq_rho;
    local pos, vel, fitness, pbest, pbestFit, j;

    z0_eps := Generate(float(range=0.0..1.0)):
    z0_rho := Generate(float(range=0.0..1.0)):

    zseq_eps := LogisticSequence(z0_eps, 3.99, NP+5):
    zseq_rho := LogisticSequence(z0_rho, 3.99, NP+5):

    pos := Array(1..NP):
    vel := Array(1..NP):
    fitness := Array(1..NP):
    pbest := Array(1..NP):
    pbestFit := Array(1..NP):

    for j from 1 to NP do
        pos[j] := Vector([evalf(eps_min + zseq_eps[j]*(eps_max -
           eps_min)),
                          evalf(rho_min + zseq_rho[j]*(rho_max -
                             rho_min))]):
        vel[j] := Vector([0.0, 0.0]):
        fitness[j] := evalf(Fitness(pos[j][1], pos[j][2])):
        pbest[j] := pos[j]:
        pbestFit[j] := fitness[j]:
    end do:

    return pos, vel, fitness, pbest, pbestFit:
end proc:

# ----------------------------
# Main PSO algorithm
# ----------------------------
PSO_Run := proc()
    local pos, vel, fitness, pbest, pbestFit;
    local gbest, gbestFit, lastGbestFit, noImproveCount;
    local r1, r2, c1, c2, w_scalar;
    local xtSeq, x0, iter, j;

    pos, vel, fitness, pbest, pbestFit := InitializeSwarmPSO(NP,
       eps_min, eps_max, rho_min, rho_max):

    gbestFit := 1e300:
    for j from 1 to NP do
        if evalf(fitness[j]) < evalf(gbestFit) then
```

```
            gbestFit := fitness[j]:
            gbest := pos[j]:
        end if:
    end do:

    lastGbestFit := gbestFit:
    noImproveCount := 0:

    x0 := Generate(float(range=0.0..1.0)):
    xtSeq := SineWeightSequence(x0, 3.9, Tmax+5):

    for iter from 1 to Tmax do
        c1 := evalf(2 + 0.5*sin(((1 - iter/Tmax)*Pi)/2)):
        c2 := evalf(2 + 0.5*cos(((1 - iter/Tmax)*Pi)/2)):

        w_scalar := xtSeq[iter]:

        for j from 1 to NP do
            r1 := Generate(float(range=0.0..1.0)):
            r2 := Generate(float(range=0.0..1.0)):

            vel[j] := evalf(w_scalar*vel[j] + c1*r1*(pbest[j]-pos[j])
                + c2*r2*(gbest-pos[j])):
            pos[j] := evalf(pos[j] + vel[j]):

            # Clamp particles to allowed range
            if pos[j][1] < eps_min then pos[j][1] := eps_min fi:
            if pos[j][1] > eps_max then pos[j][1] := eps_max fi:
            if pos[j][2] < rho_min then pos[j][2] := rho_min fi:
            if pos[j][2] > rho_max then pos[j][2] := rho_max fi:

            fitness[j] := evalf(Fitness(pos[j][1], pos[j][2])):

            if evalf(fitness[j]) < evalf(pbestFit[j]) then
                pbest[j] := pos[j]:
                pbestFit[j] := fitness[j]:
            end if:
        end do:

        for j from 1 to NP do
            if evalf(fitness[j]) < evalf(gbestFit) then
                gbestFit := fitness[j]:
                gbest := pos[j]:
            end if:
        end do:

        if evalf(abs(gbestFit - lastGbestFit)) < tol then
            noImproveCount := noImproveCount + 1:
        else
            noImproveCount := 0:
        end if:

```

```
        lastGbestFit := gbestFit:

        if noImproveCount >= noImproveLimit then
            break:
        end if:
    end do:

    return gbest, gbestFit, iter:
end proc:

# ------------------------------
# Final execution
# ------------------------------
Gbest, GbestFit, IterDone := PSO_Run():
printf("\nFinal: Iter %d, BestFit = %.12e, eps = %.6f, rho = %.6f\n",
       round(IterDone), GbestFit, Gbest[1], Gbest[2]);
```