```julia
# Check if all the packages are installed or not
cond = "Gadfly" in keys(Pkg.installed()) &&
"Colors" in keys(Pkg.installed()) &&
"ODEInterface" in keys(Pkg.installed()) &&
"ForwardDiff" in keys(Pkg.installed());
@assert cond "Please check if the following package(s) are installed:\
    Gadfly\
    Colors\
    ODEInterface\
    ForwardDiff"

# Load all the required packages
using Gadfly
using Colors
using ODEInterface
using ForwardDiff
@ODEInterface.import_huge
loadODESolvers();

# Define the right-hand function for automatic differentiation
function vdpolAD(x)
    return [x[2],((1-x[1]^2)*x[2]-x[1])*1e6]
end

# Define the system for the solver
function vdpol(t,x,dx)
    dx[:] = vdpolAD(x);
    return nothing
end

# Define the Jacobian function using AD
function getJacobian(t,x,J)
    J[:,:] = ForwardDiff.jacobian(vdpolAD,x);
    return nothing
end

# Flag to check whether plot is to be generated and saved or not
# Also checks if all solvers are successful
printFlag = true;

# Initial conditions
t0 = 0.0; T = [1.0:11.0;]; x0 = [2.0,0.0];

# Get "reference solution" from
# http://www.unige.ch/~hairer/testset/testset.html
f = open("vdpolRefSol.txt");
lines = readlines(f);

x_ref = Array{Float64}(11);

tmp = Array{Float64}(22);
counter = 1;
for l in lines
    tmp[counter] = parse(Float64,l);
    counter += 1;
end

x_ref = tmp[1:2:end];

close(f)

# Store the solver names for plotting
solverNames = ["RADAU","RADAU5","SEULEX"];

# Initialize the variables for plots
# f_e = number of function evaluations
f_e = zeros(33,3);
# err = error wrt ref solution over all time steps and components
err = zeros(33,3);
# flops = Floating point operations
flops = zeros(33,3);

# Weights for computing flops
dim = 2; # dimension of the system
flopsRHS = 5; # Counted
flopsLU = ceil(2*((dim)^3)/3); # As per LU algorithm (can be less)
flopsFW_BW = (dim)^2; # As per FW/BW algorithm (can be less)
flopsJac = ceil(1.5*flopsRHS); # A guess at the moment

for i =0:32

    # Set the tolerance for current run
    Tol = 10^(-2-i/4);

    # Set solver options
    opt = OptionsODE(OPT_EPS=>1.11e-16,OPT_ATOL=>Tol,OPT_RTOL=>Tol,
    OPT_RHS_CALLMODE => RHS_CALL_INSITU,
    OPT_JACOBIMATRIX=>getJacobian);

    # Store the stats of the last t_end
    # for computing flops
    stats = Dict{ASCIIString,Any};

    # Restart the solution for each end time
    # to ensure a more accurate solution
    # compared to dense output

    # Solve using RADAU
    x_radau = Array{Float64}(11);
    for j=1:11
        (t,x,retcode,stats) = radau(vdpol,t0, T[j], x0, opt);
        # If solver fails do not continue further
        if retcode != 1
            println("Solver RADAU failed");
            printFlag = false;
            break;
        end
        x_radau[j] = x[1];
        f_e[i+1,1] = stats.vals[13];
    end
    # If solver fails do not continue further
    if !printFlag
        break;
    end
    err[i+1,1] = norm(x_radau-x_ref,Inf);
    flops[i+1,1] = stats["no_rhs_calls"]*flopsRHS+
                   stats["no_fw_bw_subst"]*flopsFW_BW +
                   stats["no_jac_calls"]*flopsJac+
                   stats["no_lu_decomp"]*flopsLU;

    # Solve using RADAU5
    x_radau5 = Array{Float64}(11);
    for j=1:11
        (t,x,retcode,stats) = radau5(vdpol,t0, T[j], x0, opt);
        # If solver fails do not continue further
        if retcode != 1
        println("Solver RADAU5 failed");
            printFlag = false;
            break;
        end
        x_radau5[j] = x[1];
        f_e[i+1,2] = stats.vals[13];
    end
    # If solver fails do not continue further
    if !printFlag
        break;
    end
    err[i+1,2] = norm(x_radau5-x_ref,Inf);
    flops[i+1,2] = stats["no_rhs_calls"]*flopsRHS+
                   stats["no_fw_bw_subst"]*flopsFW_BW +
                   stats["no_jac_calls"]*flopsJac+
                   stats["no_lu_decomp"]*flopsLU;

    # Solve using SEULEX
    x_seulex = Array{Float64}(11);
    for j=1:11
        (t,x,retcode,stats) = seulex(vdpol,t0, T[j], x0, opt);
        # If solver fails do not continue further
        if retcode != 1
            println("Solver seulex failed");
            printFlag = false;
            break;
        end
        x_seulex[j] = x[1];
        f_e[i+1,3] = stats.vals[13];
    end
    # If solver fails do not continue further
    if !printFlag
        break;
    end
    # Get the error over all the components and
    err[i+1,3] = norm(x_seulex-x_ref,Inf);
```

```
    flops[i+1,3] = stats["no_rhs_calls"]*flopsRHS+
                   stats["no_fw_bw_subst"]*flopsFW_BW +
                   stats["no_jac_calls"]*flopsJac+
                   stats["no_lu_decomp"]*flopsLU;
end

# Save the plot in PNG format
# if all the solvers were successful
if printFlag
    savePlotPNG("vdpolPrecisionTest",f_e,err,solverNames);
else
    println("Plot cannot be generated due to failure");
end
```