NDSolve2016

In[1549]:=
```
Clear["Global`*"]
```

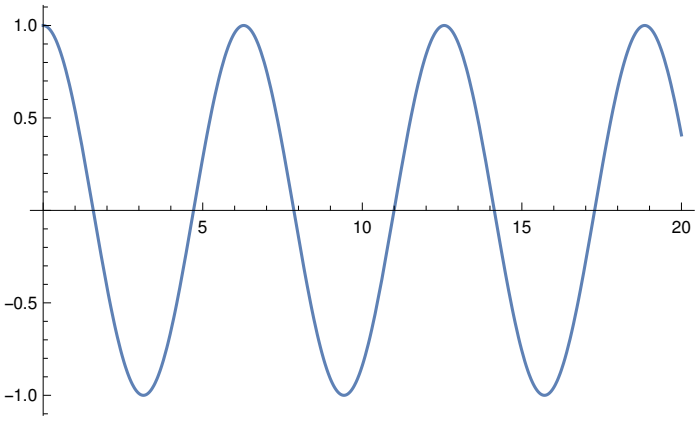
In[1550]:=
```
(* Example of NDSolve *)
```

In[1551]:=
```
tmax = 20;
nd = NDSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0},
  x[t], {t, 0, tmax}]
```

Out[1552]=
```
{{x[t] → InterpolatingFunction[ Domain: {{0., 20.}}
                                Output: scalar ][t]}}
```

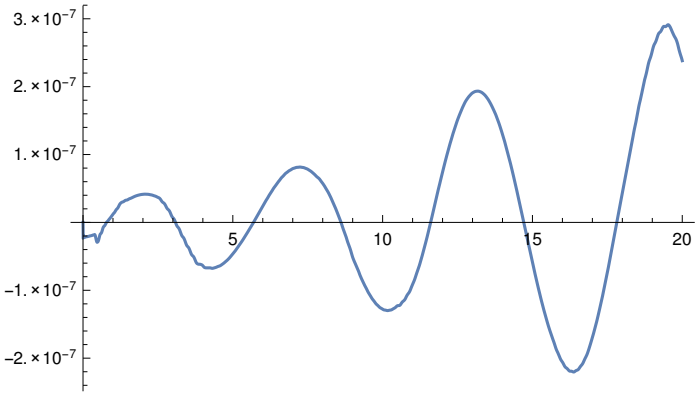
In[1553]:=
```
Plot[x[t] /. nd[[1]], {t, 0, tmax}]
```

Out[1553]=



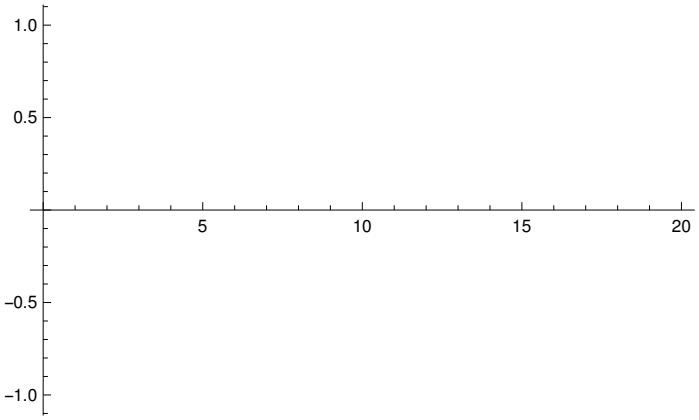
In[1554]:=
```
Plot[(x[t] /. nd[[1]]) - Cos[t], {t, 0, tmax}]
(* Compare with exact solution: error small but rising. *)
```

Out[1554]=



In[1555]:=
```
Plot[x'[t] /. nd[[1]], {t, 0, tmax}]
(* Try to plot derivative:  this way doesn't work! *)
```

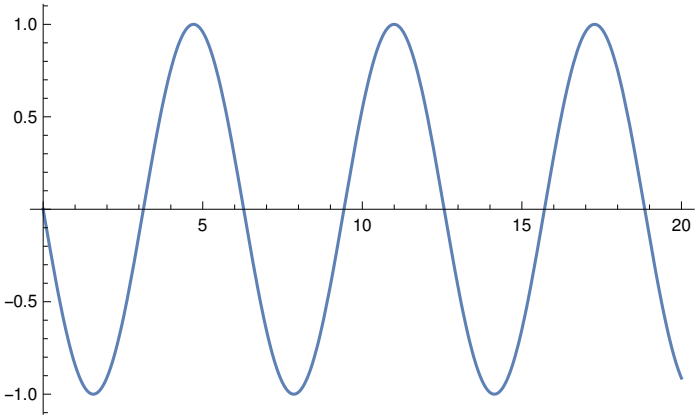
Out[1555]=



In[1556]:=
```
xx[t_] = x[t] /. nd[[1]]
Plot[xx'[t], {t, 0, tmax}]
(* Try to plot derivative:  this way DOES work! *)
```

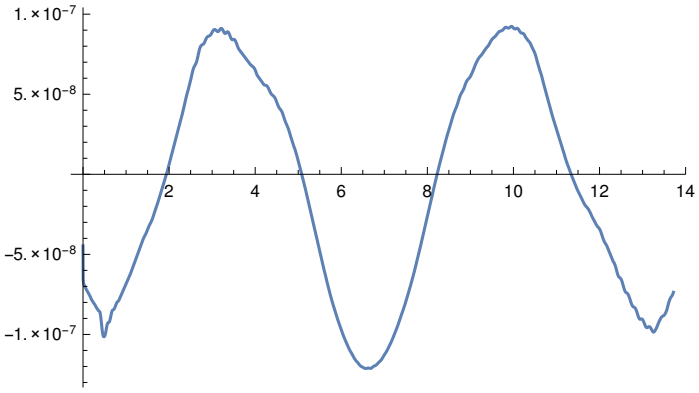
Out[1556]= InterpolatingFunction[Domain: {{0., 20.}} Output: scalar][t]

Out[1557]=



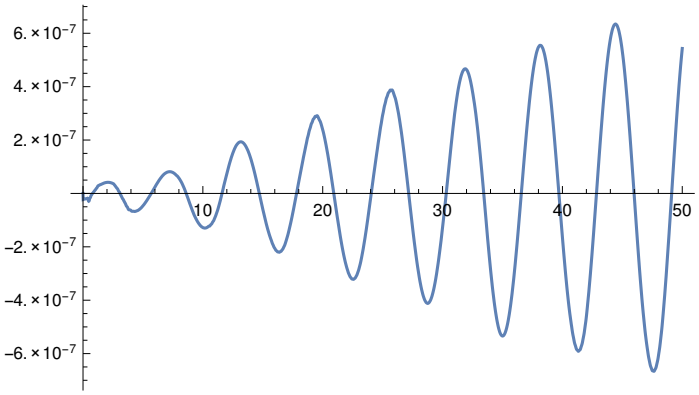
In[1558]:=
```
Plot[xx[t] - xx[t + 2 * Pi], {t, 0, tmax - 2 * Pi}]
(* Test that answer is periodic (within numerical accuracy of 10^-7) *)
```

Out[1558]=



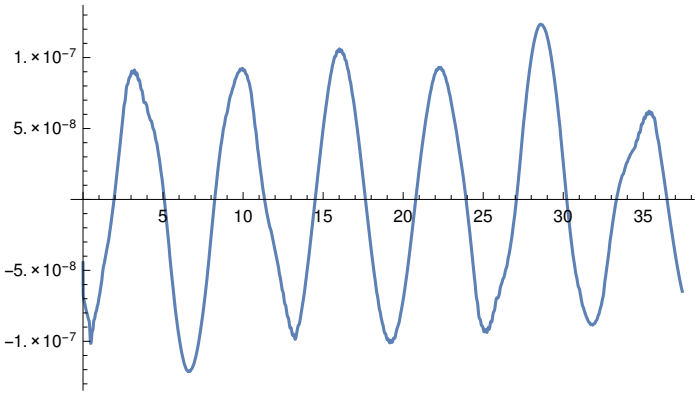
In[1559]:=
```
tmax = 50;
nd = NDSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0},
  x[t], {t, 0, tmax}];
Plot[(x[t] /. nd[[1]]) - Cos[t], {t, 0, tmax}]
xx[t_] = x[t] /. nd[[1]]
Plot[xx[t] - xx[t + 2 * Pi], {t, 0, tmax - 4 * Pi}, PlotRange → All]
(* This shows that default accuracy here is 10^-6 *)
```

Out[1561]=



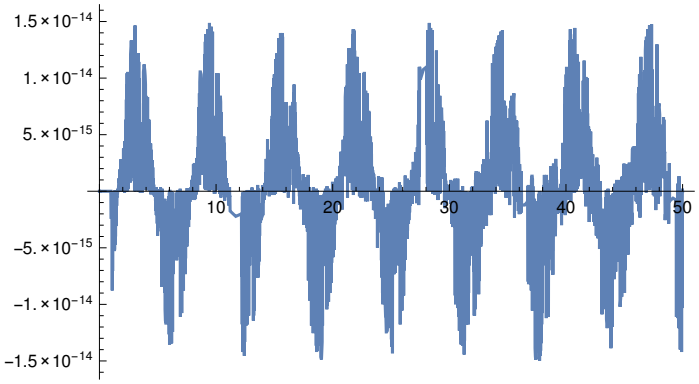
Out[1562]= InterpolatingFunction[Domain: {{0., 50.}} Output: scalar][t]

Out[1563]=



In[1564]:=
```
tmax = 50;
nd = NDSolve[{x''[t] + x[t] == 0, x[0] == 1, x'[0] == 0},
  x[t], {t, 0, tmax}, AccuracyGoal → 20,
    PrecisionGoal → 20, WorkingPrecision → 100];
Plot[(x[t] /. nd[[1]]) - Cos[t], {t, 0, tmax}, PlotRange → All]
xx[t_] = x[t] /. nd[[1]]
Plot[xx[t] - xx[t + 2 * Pi], {t, 0, tmax - 4 * Pi}, PlotRange → All]
(* This shows how accuracy can be improved *)
```

Out[1566]=



Out[1567]= InterpolatingFunction[

Domain: {{0, 50.000000000000000000000000000000000000000000000000000000000000000000000000000000}}

Output: scalar

Out[1568]=