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

Solution of PDE with periodic boundary conditions

Problem

Solve numerically the differential equation

$$\frac{\partial u(x,t)}{\partial t} = c\,\frac{\partial u(x,t)}{\partial x} \tag{1}$$

with the following initial condition

$$u(x,0) = e^{\sin(x)} \tag{2}$$

and periodic boundary conditions

$$u(0,t) = u(2\pi,t) \tag{3}$$

and compare the results with the exact solution

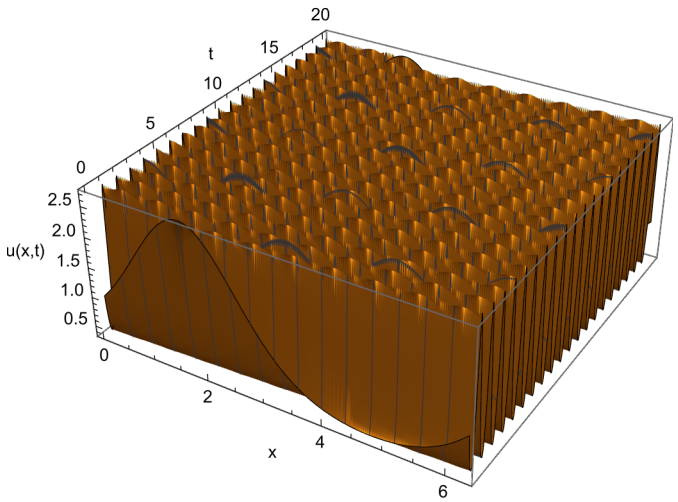
$$u(x,0) = e^{\sin(x+ct)} \tag{4}$$

Exact solution

In[ ]:=
```
{xmin, xmax} = {0, 2 π};
{tmin, tmax} = {0, 20};
c = -2 π;
u0[x_, t_] = Exp[Sin[x + c * t]];
sol = DSolve[{D[u[x, t], t] == c D[u[x, t], x], u[x, tmin] == u0[x, tmin]}, u, {x, t}]
Plot3D[u[x, t] /. sol, {x, xmin, xmax}, {t, tmin, tmax},
 PlotRange → All, PlotPoints → 100, AxesLabel → {"x", "t", "u(x,t)"}]
```

Out[ ]=
```
{{u → Function[{x, t}, e^-Sin[2 π t-x]]}}
```

Out[ ]=



Numerical solution using built-in function

In[ ]:=
```
numsol = NDSolve[{D[u[x, t], t] == c D[u[x, t], x], u[x, tmin] == u0[x, tmin],
    u[xmin, t] == u[xmax, t]}, u, {x, xmin, xmax}, {t, tmin, tmax}]
Plot3D[Evaluate[u[x, t] /. numsol], {x, xmin, xmax}, {t, tmin, tmax},
 PlotRange → All, PlotPoints → 100, AxesLabel → {"x", "t", "u(x,t)"}]
Plot3D[Evaluate[Abs[u0[x, t] - u[x, t] /. numsol]], {x, xmin, xmax},
 {t, tmin, tmax}, PlotRange → All, PlotPoints → 100, AxesLabel → {"x", "t", "error"}]
```

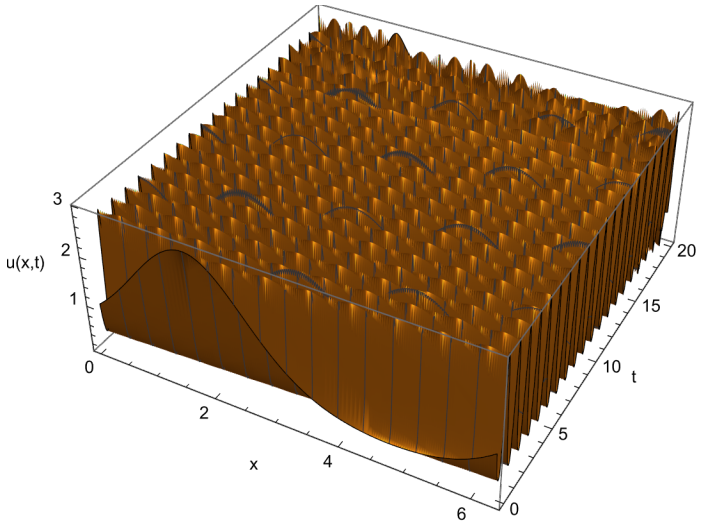
NDSolve: Warning: scaled local spatial error estimate of 703.1546502485807` at t = 20.` in the direction of independent variable x is much greater than the prescribed error tolerance. Grid spacing with 49 points may be too large to achieve the desired accuracy or precision. A singularity may have formed or a smaller grid spacing can be specified using the MaxStepSize or MinPoints method options.

Out[ ]=

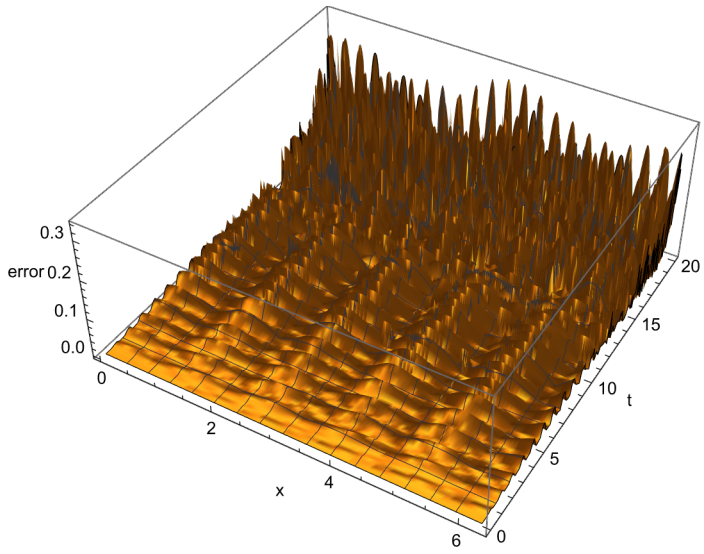
```
{{u → InterpolatingFunction[ Domain: {{0., 6.28}, {0., 20.}}
                             Output: scalar ]}}
```

Data not in notebook. Store now

Out[ ]=



Out[ ]=



Auxiliary functions

Construction of the differentiation matrices for the finite-difference and spectral methods

In[ ]:=

```
getFDMatrix[n_, p_, h_] := Module[{v, pf2},
   v = ConstantArray[0, n];
   pf2 = Factorial[p]^2;
   Do[
    v[[s + 1]] = (-1)^(s + 1) * pf2 / (s * Factorial[p - s] * Factorial[p + s]);
    v[[n - s + 1]] = -v[[s + 1]],
    {s, 1, p}
   ];
   Return[ToeplitzMatrix[v, -v] / h]
  ];

getSpectralMatrix[n_, h_] := Module[{v},
   If[Mod[n, 2] == 0,
    v = Flatten[{0, Table[0.5 * (-1)^i * Cot[i * h / 2], {i, 1, n - 1}]}],
    (* even number of points *)
    v = Flatten[{0, Table[0.5 * (-1)^i * Csc[i * h / 2], {i, 1, n - 1}]}]
    (* odd number of points *)
   ];
   Return[ToeplitzMatrix[v, -v]];
  ];
```

Functions to run the Runge-Kutta methods for time propagation

In[ ]:=

```
(* predefined coeefficients of some explicit Runge-Kutta methods*)
MyGetRKCoefficients[method_, s_] :=
```

```
 Module[{c, A, b},
 MyGetRKCoefficients::method = "Unknown method `1`. Classical R-K method returned.";
 c = ConstantArray[0, s];
 A = ConstantArray[0, {s, s}];
 b = ConstantArray[0, s];
 Switch[method,
  "Classical",
   Switch[s,
    1,
     c = {0};
     A = {{0}};
     b = {1},
    2,
     c = {0, 1 / 2};
     A = {{0, 0}, {1 / 2, 0}};
     b = {0, 1},
    4,
     c = {0, 1 / 2, 1 / 2, 1};
     A = {{0, 0, 0, 0}, {1 / 2, 0, 0, 0}, {0, 1 / 2, 0, 0}, {0, 0, 1, 0}};
     b = {1 / 6, 1 / 3, 1 / 3, 1 / 6},
    _,
     c = {0, 1 / 2};
     A = {{0, 0}, {1 / 2, 0}};
     b = {0, 1}
   ],
  "Cash-Karp",
   c = {0, 1 / 5, 3 / 10, 3 / 5, 1, 7 / 8};
   A = {{0, 0, 0, 0, 0, 0},
     {1 / 5, 0, 0, 0, 0, 0},
     {3 / 40, 9 / 40, 0, 0, 0, 0},
     {3 / 10, -9 / 10, 6 / 5, 0, 0, 0},
     {-11 / 54, 5 / 2, -70 / 27, 35 / 27, 0, 0},
     {1631 / 55 296, 175 / 512, 575 / 13 824, 44 275 / 110 592, 253 / 4096, 0}};
   Switch[s,
    4,
     b = {2825 / 27 648, 0, 18 575 / 48 384, 13 525 / 55 296, 277 / 14 336, 1 / 4},
    _, (* standard is this method of the order 5 *)
     b = {37 / 378, 0, 250 / 621, 125 / 594, 0, 512 / 1771}
   ],
  _,
   Message[MyGetRKCoefficients::method, method];
   {c, A, b} = MyGetRKCoefficients["Classical", s]
 ];
 Return[{c, A, b}]
];

(* main function for solution of ODEs using the explicit Runge-Kutta methods *)
```

```
MyODESolveUsingRK[{cs_, As_, bs_}, f_, t0_, vini_, k_, n_] :=
 Module[{c, A, b, r, t, v, y, i, j, l, nvar},
 c = N[cs]; A = N[As]; b = N[bs];
 r = Length[c];  (* number of intersteps *)
 t = N[t0 + k Range[0, n]];
 nvar = Length[vini];
 If[nvar == 0,
  v = ConstantArray[0.0, n + 1];
  y = ConstantArray[0.0, r],
  v = ConstantArray[0.0, {n + 1, nvar}];
  y = ConstantArray[0.0, {r, nvar}]
 ];
 v[[1]] = N[vini];
 (* only explicit method is implemented yet *)
 Do[
  Do[
   y[[l]] = v[[i]] + k * Sum[A[[l]][[j]] * f[y[[j]], t[[i]] + c[[j]] * k], {j, 1, r}],
   {l, 1, r}
  ];
  v[[i + 1]] = v[[i]] + k * Sum[b[[j]] * f[y[[j]], t[[i]] + c[[j]] * k], {j, 1, r}],
  {i, 1, n}
 ];
 Return[Transpose[{t, v}]]
];
```

Main function to solve the given problem

In[ ]:=
```
(* parameters of solvePeriodicWave are
      nx = number of space grid points (xmin and xmax taken from above),
      nt = number of time steps (tmin and tmax taken from above),
      m  = differentiation matrix to use for space derivative
           (1 = spectral, 2 = FD2, 3 = FD4, 4 = FD6, etc),
      RK = Runge-Kutta method to use ("Classical" or "Cash-Karp"),
      RKorder = order of the Runge-Kutta method
                (1,2,4 for "Classical", 4,5 for "Cash-Karp"),
      printλ = 0 or 1 whether to print λ = c dt/dx
               (should be less or equal 1 for "Classical", 0.8 for "Cash-Karp"),
      saveFullSolution =
 0 or 1 whether the full solution is returned or only at time tmax
*)
solvePeriodicWave[nx_, nt_, m_, RK_, RKorder_, printλ_, saveFullSolution_] :=
  Module[{dx, dt, X, T, vini, vfin, tmp, DM, f, v, sol, error},
   (* grid initialization *)
   dx = (xmax - xmin) / nx;
   dt = (tmax - tmin) / nt;
   If[printλ == 1, Print["λ = ", c N[dt / dx]]];
   X = N[Range[xmin, xmax - dx, dx]];
   T = N[Range[tmin, tmax, dt]];

   (* initial and final state on the grid *)
   vini = u0[X, tmin];
   vfin = u0[X, tmax];
   (* construction of the differentiation matrix *)
   If[m == 1, (* spectral method *)
    DM = getSpectralMatrix[nx, dx],
    (* finite-difference methods *)
    DM = getFDMatrix[nx, m - 1, dx]
   ];
   (* solution of PDE using the Runge-Kutta method *)
   f[u_, t_] := c * DM.u;
   v = MyODESolveUsingRK[MyGetRKCoefficients[RK, RKorder], f, tmin, vini, dt, nt];
   If[saveFullSolution == 1,
    (* save full solution *)
    sol = v;
    error = Abs[v[[All, 2]] - Map[u0[X, #] &, T]],
    (* save solution at tmax*)
    sol = v[[nt + 1, 2]];
    error = Abs[sol - vfin]
   ];
   Return[{X, sol, error}]
  ];
```

Numerical solution using differentiation matrices to approximate space

derivatives and the 4th-order Runge-Kutta method for time propagation

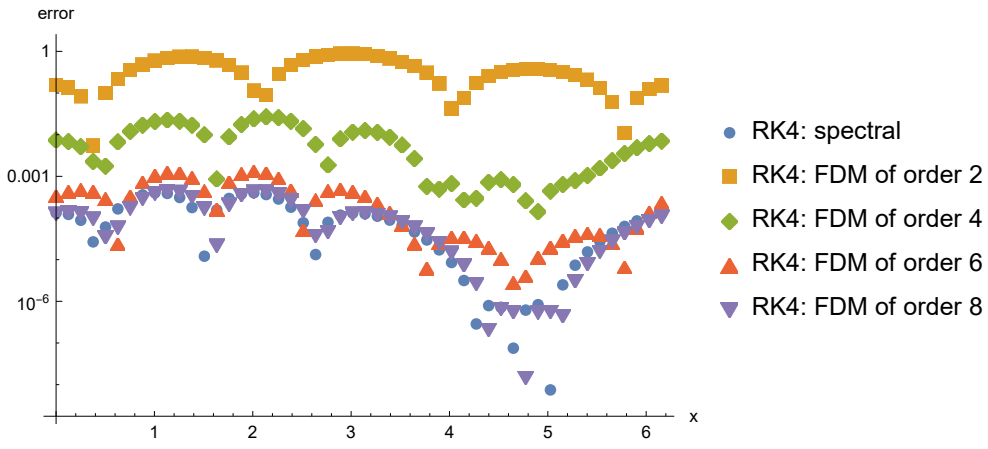
In[ ]:=
```
{nx, nt} = {50, 2000};
nm = 5;
solRK4 = ConstantArray[0.0, nm];
errorRK4 = ConstantArray[0.0, nm];

Do[ (* loop over methods - the first is spectral method,
  then FDMs of increasing even order *)
  {XRK4, solRK4[[m]], errorRK4[[m]]} = solvePeriodicWave[nx, nt, m, "Classical", 4, 1, 0],
  {m, 1, nm} (* loop over 4 methods *)
 ];
```

```
λ = -0.5
λ = -0.5
λ = -0.5
λ = -0.5
λ = -0.5
```

In[ ]:=
```
plot1 =
 ListLogPlot[Table[Transpose[{XRK4, errorRK4[[m]]}], {m, 1, nm}], PlotMarkers → Automatic,
  PlotLegends → {"RK4: spectral", "RK4: FDM of order 2", "RK4: FDM of order 4",
    "RK4: FDM of order 6", "RK4: FDM of order 8"}, AxesLabel → {"x", "error"}]
```

Out[ ]=



Phase error analysis - dispersion relations

In[ ]:=
```
ck[m_, k_, h_] := Module[{mf2, alpha, sum},
   mf2 = Factorial[m]^2;
   sum = 0;
   Do[
    alpha = (-1)^(s + 1) * 2 * mf2 / (Factorial[m - s] * Factorial[m + s]);
    sum = sum + alpha * Sin[s * k * h] / (k * h * s),
    {s, 1, m}
   ];
   Return[c * sum]
  ];
capprox[m_, k_, h_] := Normal[Series[ck[m, k, h], {k, 0, 2 m}]];
hh = 1 / 5;
Do[
  Print[ck[m, k, hh]];
  Print[capprox[m, k, hh]],
  {m, 1, 3}
 ];
Plot[Evaluate[Flatten[{c, Table[{ck[m, k, hh], capprox[m, k, hh]}, {m, 1, 3}]}]],
 {k, -20, 20}, PlotRange → {-7, 2}]
```

$$-\frac{10\,\pi\,\text{Sin}\left[\frac{k}{5}\right]}{k}$$

$$-2\,\pi+\frac{k^2\,\pi}{75}$$

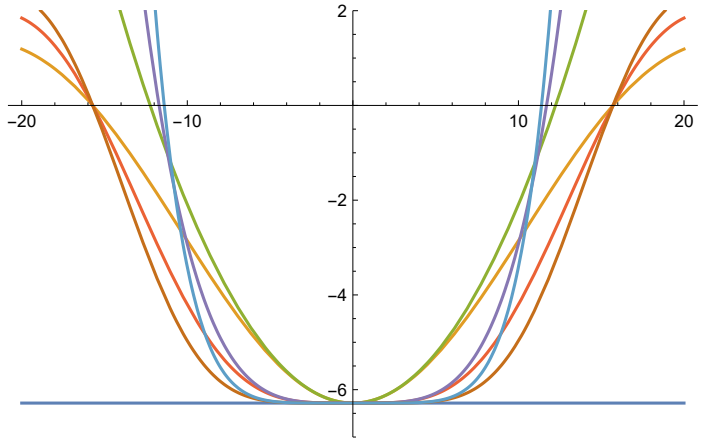
$$-2\,\pi\left(\frac{20\,\text{Sin}\left[\frac{k}{5}\right]}{3\,k}-\frac{5\,\text{Sin}\left[\frac{2\,k}{5}\right]}{6\,k}\right)$$

$$-2\,\pi+\frac{k^4\,\pi}{9375}$$

$$-2\,\pi\left(\frac{15\,\text{Sin}\left[\frac{k}{5}\right]}{2\,k}-\frac{3\,\text{Sin}\left[\frac{2\,k}{5}\right]}{2\,k}+\frac{\text{Sin}\left[\frac{3\,k}{5}\right]}{6\,k}\right)$$

$$-2\,\pi+\frac{k^6\,\pi}{1\,093\,750}$$

Out[ ]=



Numerical solution using differentiation matrices to approximate space derivatives and the Cash-Karp (Runge-Kutta of order 5) method for time propagation

In[ ]:=

```
{nx, nt} = {50, 2000};
nm = 5;
solCK5 = ConstantArray[0.0, nm];
errorCK5 = ConstantArray[0.0, nm];

Do[ (* loop over methods - the first is spectral method,
  then FDMs of increasing even order *)
  {XCK5, solCK5[[m]], errorCK5[[m]]} = solvePeriodicWave[nx, nt, m, "Cash-Karp", 5, 1, 0],
  {m, 1, nm} (* loop over 4 methods *)
 ];
```

λ = -0.5

λ = -0.5

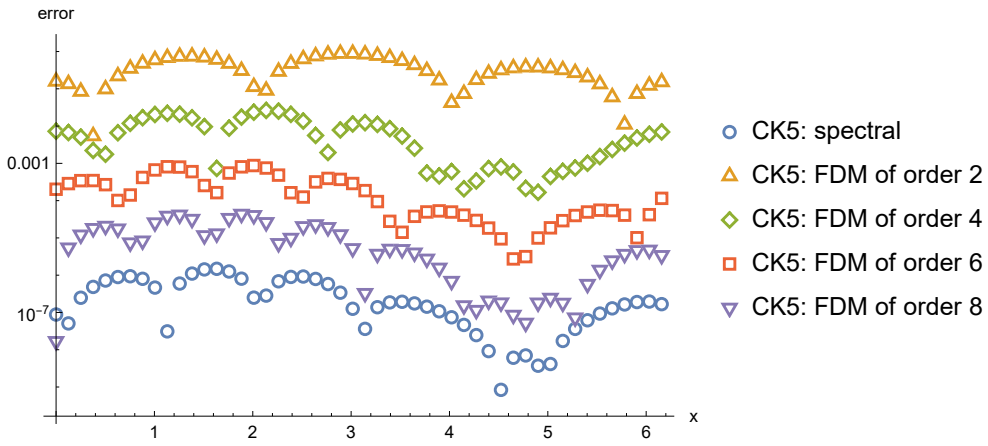
λ = -0.5

λ = -0.5

λ = -0.5

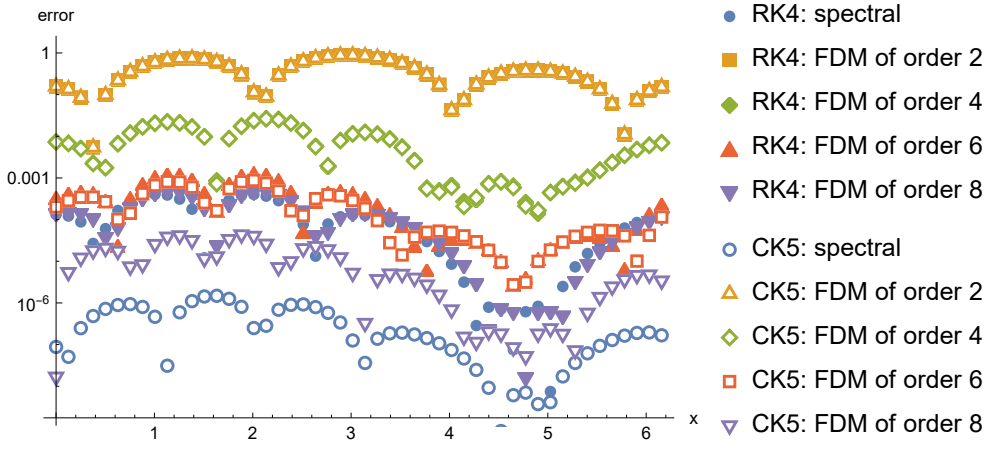
In[ ]:=
```
plot2 = ListLogPlot[
  Table[Transpose[{XCK5, errorCK5[[m]]}], {m, 1, nm}], PlotMarkers → "OpenMarkers",
  PlotLegends → {"CK5: spectral", "CK5: FDM of order 2", "CK5: FDM of order 4",
    "CK5: FDM of order 6", "CK5: FDM of order 8"}, AxesLabel → {"x", "error"}]
```

Out[ ]=



In[ ]:=
```
Show[{plot1, plot2}]
```

Out[ ]=



```
nx = Range[8, 20, 4];
nxs = Length[nx];
nt = {100, 500, 1000, 5000, 10000, 50000};
nts = Length[nt];
timeS = ConstantArray[0.0, {nxs, nts}];
maxErrorS = ConstantArray[0.0, {nxs, nts}];
Do[
  (* parameters of solvePeriodicWave are nx ,
  nt, and method (1 = spectral, 2 = FD2, 3 = FD4, 4 = FD6, ) *)
  {timeS[[ix, it]], {XS, solS, errorS}} =
   Timing[solvePeriodicWave[nx[[ix]], nt[[it]], 1, "Cash-Karp", 5, 1, 0]];
  maxErrorS[[ix, it]] = If[Max[errorS] > 10, 10, Max[errorS]],
  {it, 1, nts}, {ix, nxs}
 ];
```

λ = -1.6

λ = -2.4

λ = -3.2

λ = -4.

λ = -0.32

λ = -0.48

λ = -0.64

λ = -0.8

λ = -0.16

λ = -0.24

λ = -0.32

λ = -0.4

λ = -0.032

λ = -0.048

λ = -0.064

λ = -0.08

λ = -0.016

λ = -0.024

λ = -0.032

λ = -0.04

λ = -0.0032

λ = -0.0048

λ = -0.0064

λ = -0.008

In[ ]:=
```
timeS
maxErrorS
```

Out[ ]=
```
{{0.03125, 0.125, 0.25, 1.109375, 2.203125, 11.15625},
 {0.03125, 0.109375, 0.21875, 1.125, 2.28125, 11.390625},
 {0.03125, 0.125, 0.234375, 1.15625, 2.328125, 11.671875},
 {0.09375, 0.125, 0.25, 1.1875, 2.390625, 12.15625}}
```

Out[ ]=
$$\{\{10, 0.00118147574867, 0.0000303366068732, 8.99563845635\times10^{-9}, 2.8037483446\times10^{-10}, 2.46913600677\times10^{-13}\}, \{10, 0.00222231005402, 0.000050597217331, 1.43907787908\times10^{-8}, 4.47903936163\times10^{-10}, 2.03836947321\times10^{-13}\}, \{10, 0.0023110863616, 0.0000525084192353, 1.48229064578\times10^{-8}, 4.61233273796\times10^{-10}, 1.42108547152\times10^{-13}\}, \{10, 10, 0.0000525673343073, 1.48331404937\times10^{-8}, 4.61529481299\times10^{-10}, 1.48547840695\times10^{-13}\}\}$$

In[ ]:=
```
Print["Maximal error: ", Max[errorS]];
ListLogPlot[Transpose[{XS, errorS}]]
```

```
Maximal error: 1.48547840695×10^-13
```

Out[ ]=