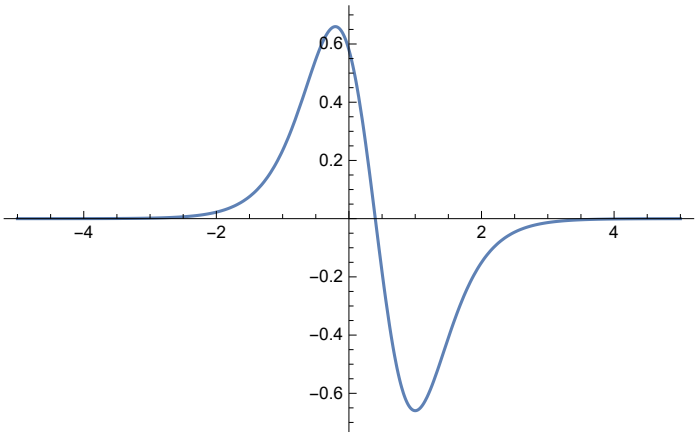
In[367]:=
```
Style["First we see the NDSolve solution", 20]
```
estilo | primeiro | resolve numéricamente equaçã

Out[367]=

First we see the NDSolve solution

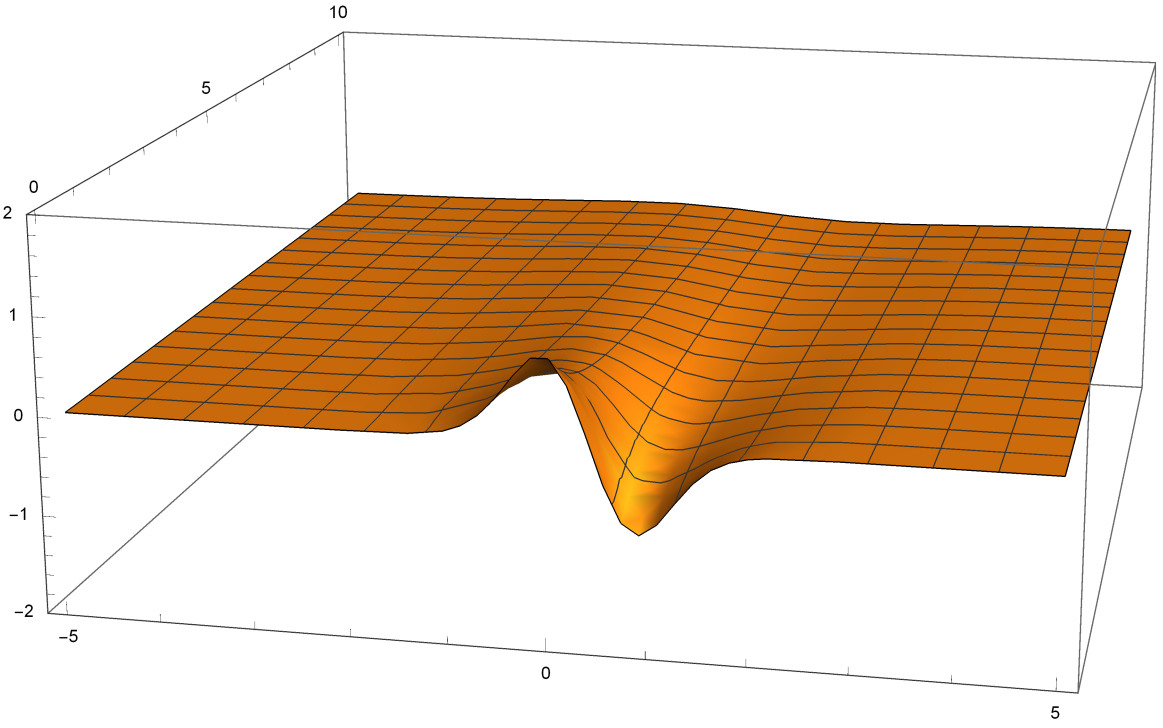
In[368]:=
```
Clear[eq, y, t, u, wave, ci, cc, dx, dt, x, k, tf]
```
apaga
```
(*Defining variables*)
wave = {};(*the inicial wave form as a sum of independent waves*)
xi = -5; (*spacial range*)
xf = 5;
ti = 0; (*temporal range*)
tf = 10;
u = 0.1; (*parameters of the equation*)
k = 0.5;
dx = 0.08; (*parameters of the solution*)
dt = dx^2 / 2;
Do[
```
repete
```
 AppendTo[wave, Sech[(2 π * x) / xf]^(2 n) - Sech[(2 π * x - xf) / xf]^(2 n)]
```
adiciona a
```
 , {n, 1, 10}] (*Constructing the waveform*)
(*Solving the equation*)
eq := D[y[x, t], t] + k * y[x, t] * D[y[x, t], x] == u * D[y[x, t], {x, 2}];
```
derivada | derivada | derivada
```
ci := y[x, ti] == wave[[2]];
cc := y[xi, t] == y[xf, t];
sis = {eq, ci, cc};
Plot[wave[[1]], {x, xi, xf}, PlotRange → All]
```
gráfico | intervalo do g··· | tudo
```
NDSolve[sis, y, {x, xi, xf}, {t, ti, tf}, MaxSteps → 100 000, MaxStepSize → {dx, dt}]
```
resolve numéricamente equação diferencial | número máximo de pa··· | tamanho máximo de passo
```
sol = y /. %[[1]];
Plot3D[sol[x, t], {x, xi, xf}, {t, ti, tf}, PlotRange → {-2, 2}, ImageSize → 600]
```
gráfico 3D | intervalo do gráfico | tamanho da imagem
```
an1 = Animate[
```
anima
```
  Graphics[Plot[sol[x, t], {x, xi, xf}, ImageSize → 600, PlotRange → 1]], {t, ti, tf}]
```
represent··· | gráfico | tamanho da imagem | intervalo do gráfico
```
t = tf;
im2 =
  Plot[sol[x, t], {x, xi, xf}, ImageSize → 600, PlotRange → All, AxesOrigin → {0, 0}];
```
gráfico | tamanho da imagem | intervalo do g··· | tudo | origem dos eixos
```
t = ti;
im2i =
  Plot[sol[x, t], {x, xi, xf}, ImageSize → 600, PlotRange → All, AxesOrigin → {0, 0}];
```
gráfico | tamanho da imagem | intervalo do g··· | tudo | origem dos eixos

Out[379]=

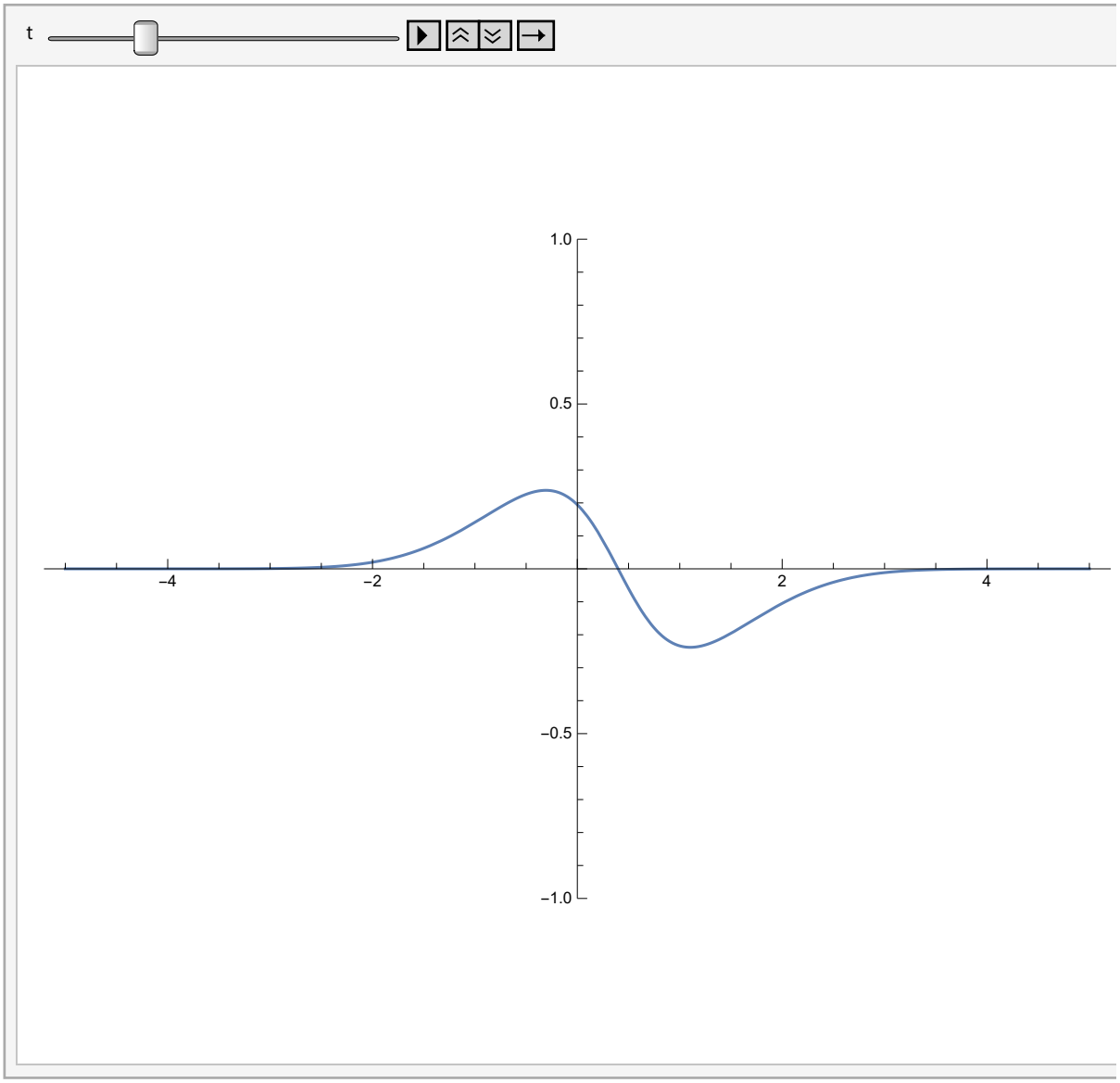


Out[380]= {{y → InterpolatingFunction[Domain: {{−5., 5.}, {0., 10.}} Output: scalar]}}

Out[382]=



Out[383]=



In[388]:=

In[389]:=
```
Style["Solving with Runge Kutta method(we
estilo
    can choose between the first 4 orders of the method)", 20]
```

Out[389]=

Solving with Runge Kutta method(we can choose between the first 4 orders of the method)

In[390]:=

In[391]:=
```
Clear[f, x, te, y, yy, y0, t, Y, M, ff, fj, NN, w, Y0, xn, tn, X, T, RK, Eqn, nx,
apaga
 a, b, tf, dt, F, a1, a2, tff, krk, krk1, krk2, MM, krk3, krk4, XX, aa1, aa2, W]
(*defining variables*)
X = {};
T = {};
NN = 50;(*number of temporal and spatial partitions*)
```

```
xn := (xf - xi) / (NN - 1);(*spatial step*)
tn := (tf - ti) / (NN - 1);(*temporal step*)
aa1 = 0.5; (*parameters of the method in all orders*)
aa2 = 1 - aa1;
a1 = 2.5 / 6;
a2 = 3 / 6;
a3 = 1 - a1 - a2;
p = 0.6;
q = 1 - p;
r = 3;
s = 8;
nx = 0;
ord = 1;(*the order of the method*)

xi = -5; (*spatial range*)
xf = 5;
ti = 0; (*temporal range*)
tf = 10;
u = 0.1;
k = 0.1;
dx = 0.08;
dt = dx^2 / 2;
w = Y0;(*inicial state*)
W = {w}; (*the solution*)
dt = (tf - ti) / (NN - 1.);
tn = dt;
t = ti;
X = Table[x, {x, xi, xf, xn}];
    tabela
T = Table[t, {t, ti, tf, tn}];
    tabela


f[x_, t_] := u * D[y[x, t], {x, 2}] - k * y[x, t] * D[y[x, t], x];
               derivada                              derivada
(*the burguers equation*)

M = {};
fj = {};
t = ti;
Do[y0[x] = wave[[1]], {x, xi, xf, xn}];
repete
Y0 = Table[y0[x], {x, xi, xf, xn}];
     tabela
Y = Y0;
Clear[t];
apaga
Y = Table[y[x, t], {x, xi, xf, xn}];
    tabela

(*Runge Kutta method in 4 orders*)
F[t_, w_, dt_] := Module[{},
                  módulo de código
    (M.w)
```

```
  ];
RK[1][F_, t_, w_, dt_] := Module[{},
                          módulo de código
   krk = F[t, w, dt];
   w + krk * dt

  ];
RK[2][F_, t_, w_, dt_] := Module[{},
                          módulo de código
   krk1 = F[t, w, dt / 2];

   krk2 = F[t + dt, w + krk1 * dt, dt / 2];

   w + (aa1 * krk1 + aa2 * krk2) * dt
];
RK[3][F_, t_, w_, dt_] := Module[{},
                          módulo de código

   krk1 = F[t, w, dt];
   krk2 = F[t + dt, w + krk1 * dt / 2, dt];
   krk3 = F[t + r * dt, w + s * krk2 * dt / 2 + (r - s) * krk1 * dt / 2, dt];
   w + (a1 * krk1 + a2 * krk2 + a3 * krk3) * dt
  ];
RK[4][F_, t_, w_, dt_] := Module[{},
                          módulo de código
   krk1 = F[t, w, dt];
   krk2 = F[t + dt, w + krk1 * dt, dt];
   krk3 = F[t + dt, w + krk2 * dt, dt];
   krk4 = F[t + dt, w + krk3 * dt, dt];
   w + (krk1 + 2 * krk2 + 2 krk3 + krk4) * dt / 6];



(*Constructing the matrix representing the temporal derivative*)


MM = {};
t = ti;
Do[
repete
  M = Table[0, {i, 1, NN}, {j, 1, NN}];
      tabela
  Do[
  repete
   Do[
   repete
     If[i == j, M[[i, j]] = -2 * u / xn^2];
     se
     If[i == j + 1, M[[i, j]] = u / xn^2 - (-k / (2 * dx)) * w[[j]]];
     se
     If[i == j - 1, M[[i, j]] = u / xn^2 + (-k / (2 * dx)) * w[[j]]];
     se
     If[i == NN && j == 1, M[[i, j]] = u / xn^2 + (-k / (2 * dx)) * w[[j]]];
     se
```

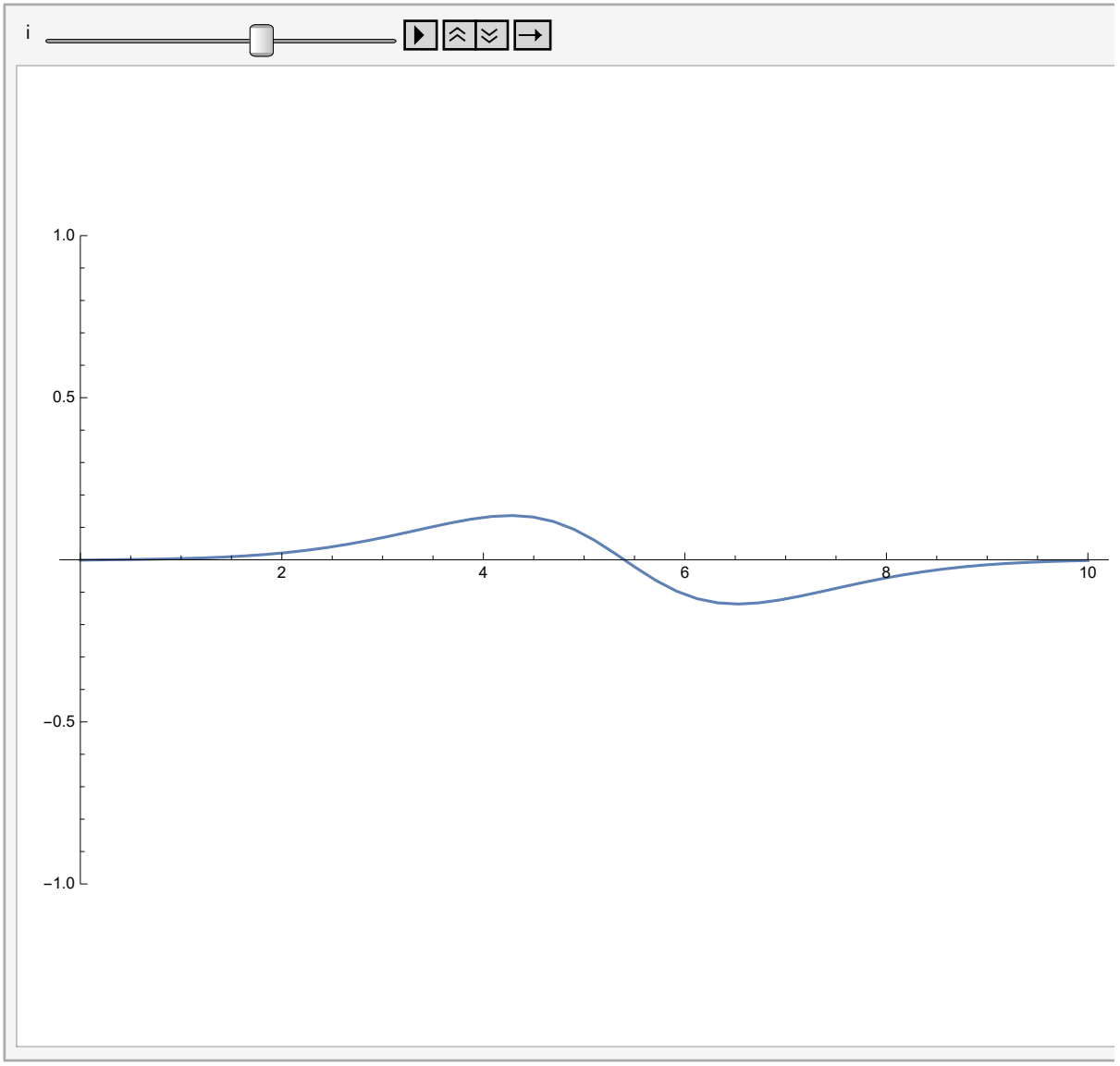
```
    If[i == 1 && j == NN, M[[i, j]] = u / xn^2 - (-k / (2 * dx)) * w[[j]]];
    se
    , {i, 1, NN}];
  , {j, 1, NN}];


w = RK[ord][F, t, w, dt];
      t = t + dt;
AppendTo[W, w];
adiciona a
AppendTo[MM, M]
adiciona a
     , {n, 2, NN}];
```

In[435]:=

In[436]:=

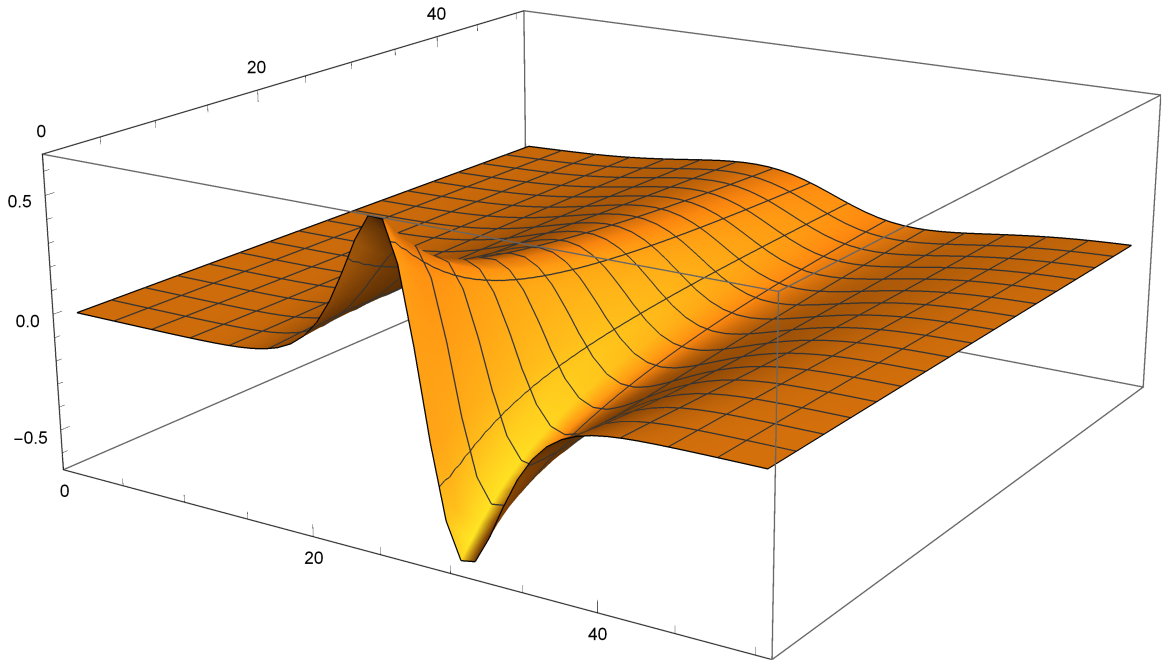
```
an2 = Animate[
        anima
  ListPlot[W[[i]], Joined → True, DataRange → xf - xi,
  gráfico de uma lista d··· unido  verd··· intervalo de dados
    PlotRange → {-1, 1}, ImageSize → 600], {i, 1, NN, 1}]
    intervalo do gráfico  tamanho da imagem
```

Out[436]=



In[437]:=
```
ListPlot3D[W, PlotRange → All, ImageSize → 600]
```
gráfico 3D de u··· | intervalo do g··· | tudo | tamanho da imagem

Out[437]=



In[440]:=
```
Export["NDSolve sol.avi", an1]
```
exporta | resolve numéricamente equação difer

```
Export["Runge Kutta sol.avi", an2]
```
exporta

Out[440]= NDSolve sol.avi

Out[441]= Runge Kutta sol.avi

In[442]:=
```
SystemOpen[DirectoryName[AbsoluteFileName["Runge Kutta sol.avi"]]]
```
abre no sist··· | nome do diretório | nome absoluto de arquivo