solving ODEs and PDEs

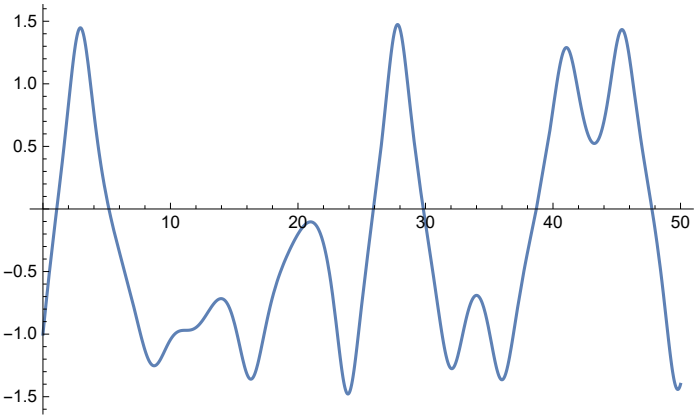
ODEs

```
In[•]:= sol = NDSolve[
          {x''[t] + 0.15 x'[t] - x[t] + x[t]^3 == 0.3 Cos[t], x[0] == -1, x'[0] == 1}, x, {t, 0, 50}]
```

```
Out[•]= {{x → InterpolatingFunction[ ⊞  Domain: {{0., 50.}}
                                     Output: scalar ]}}
```

```
In[•]:= Plot[Evaluate[x[t] /. sol[[1]]], {t, 0, 50}]
```

```
Out[•]=
```



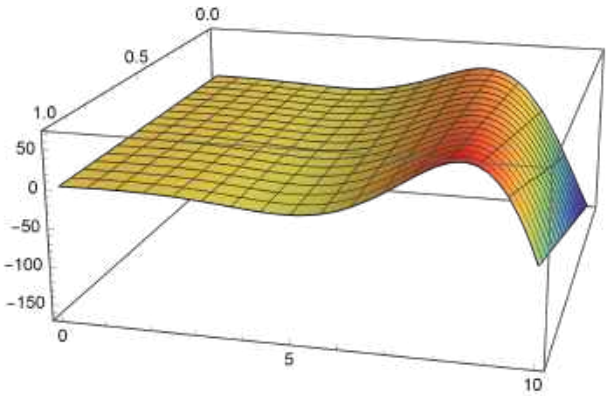

```
(* another example *)
```

```
In[•]:= sol = ParametricNDSolve[{x''[t] - x'[t] + x[t] == 0, x[0] == 1, x'[0] == s}, x, {t, 0, 10}, {s}]
```

```
Out[•]= {x → ParametricFunction[ ⊞  Expression: x
                                 Parameters: {s} ]}
```

In[•]:=
```
Plot3D[Evaluate[x[s][t] /. sol], {s, 0, 1}, {t, 0, 10},
 ColorFunction → "Rainbow", PlotPoints → 25, PlotRange → All]
```

Out[•]=




```
(* find value of s for which x → 0 at t = 10 *)
```

In[•]:=
```
root = FindRoot[Evaluate[x[s][10] /. sol ] == 0, {s, 1.}]
```

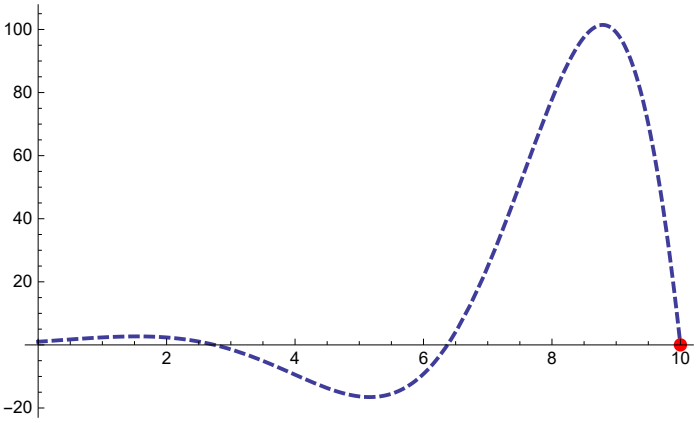
Out[•]= $\{s \to 1.40296\}$

In[•]:=
```
val = x[s /. root][10] /. sol[[1]]
```

Out[•]= -2.13635×10^{-13}

In[•]:=
```
Show[Plot[Evaluate[x[s /. root][t] /. sol[[1]]],
  {t, 0, 10}, PlotStyle → {Dashed, Thick, ColorData[1][1]}],
 Graphics[{PointSize[0.02], Red, Point[{10, val}]}]]
```

Out[•]=



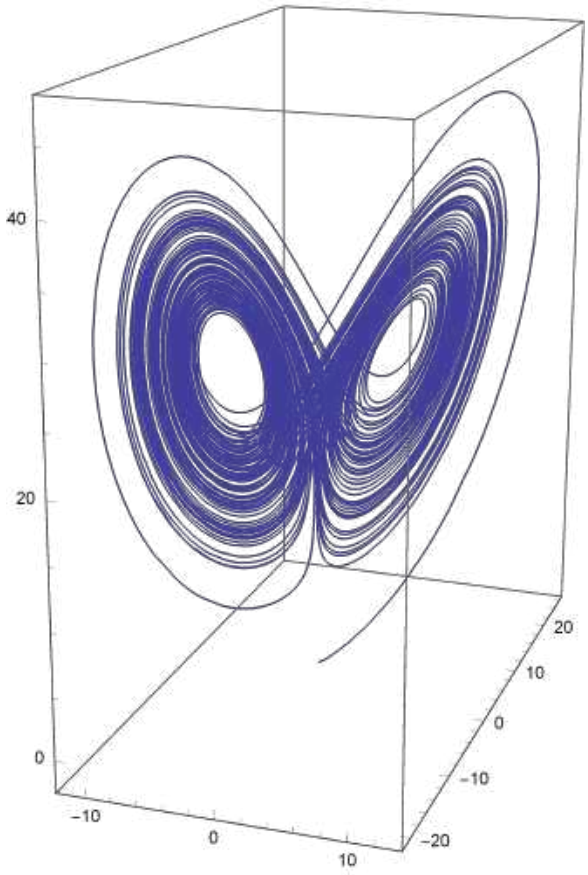
Lorenz equation

In[ ]:=
```
lsol = NDSolve[
  {x'[t] == -3 (x[t] - y[t]),
   y'[t] == -x[t] × z[t] + 26.5 x[t] - y[t], z'[t] == x[t] × y[t] - z[t],
   x[0] == z[0] == 0, y[0] == 1},
  {x, y, z}, {t, 0, 200}, MaxSteps → ∞]
```

Out[ ]=
```
{{x → InterpolatingFunction[ ⊞ Domain: {{0., 200.}} Output: scalar ],
  y → InterpolatingFunction[ ⊞ Domain: {{0., 200.}} Output: scalar ],
  z → InterpolatingFunction[ ⊞ Domain: {{0., 200.}} Output: scalar ]}}
```

In[ ]:=
```
ParametricPlot3D[Evaluate[{x[t], y[t], z[t]} /. lsol], {t, 0, 200},
 PlotPoints → 350, PlotStyle → {Thickness[0.0025], ColorData[1][1]}]
```

Out[ ]=



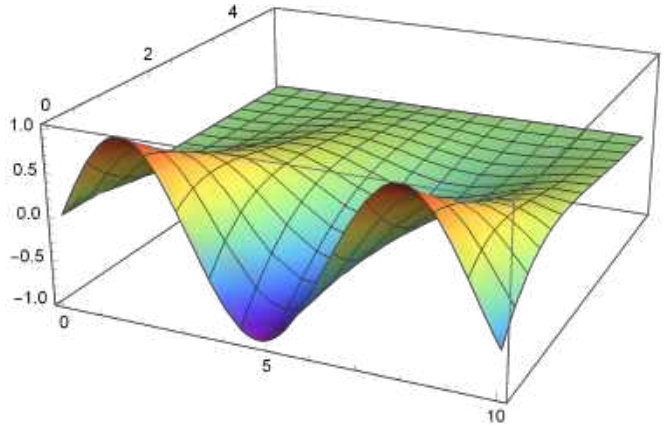
Transient Diffusion equation

```
In[•]:= hsol = NDSolve[{D[u[t, x], t] == D[u[t, x], x, x],
    u[0, x] == 0, u[t, 0] == Sin[t], u[t, 5] == 0}, u, {t, 0, 10}, {x, 0, 5}]
```

```
Out[•]= {{u → InterpolatingFunction[ ⊞ Domain: {{0., 10.}, {0., 5.}}
                                     Output: scalar ]}}
```

```
In[•]:= Plot3D[Evaluate[u[t, x] /. hsol[[1]]], {t, 0, 10},
    {x, 0, 5}, PlotRange → All, ColorFunction → "Rainbow", Mesh → Full]
```

Out[•]=



Tsunami wave (shallow water equations)

The Shallow Water Wave Equations

```
In[•]:= swe = {∂t h[λ, ϕ, t] + 1/(R Cos[ϕ]) (∂λ (u[λ, ϕ, t] (h[λ, ϕ, t] - b[λ, ϕ])) +
      ∂ϕ (v[λ, ϕ, t] (h[λ, ϕ, t] - b[λ, ϕ]) Cos[ϕ])) == 0,
    (∂ϕ u[λ, ϕ, t] × v[λ, ϕ, t])/R + (g ∂λ h[λ, ϕ, t])/(R Cos[ϕ]) + ∂t u[λ, ϕ, t] +
      (u[λ, ϕ, t] × ∂λ u[λ, ϕ, t])/(R Cos[ϕ]) == f v[λ, ϕ, t] ,
    (∂ϕ v[λ, ϕ, t] × v[λ, ϕ, t])/R + (g ∂ϕ h[λ, ϕ, t])/R + ∂t v[λ, ϕ, t] +
      (u[λ, ϕ, t] × ∂λ v[λ, ϕ, t])/(R Cos[ϕ]) == -f u[λ, ϕ, t] };
```

Shapes for Ocean Floor and Fault Displacement

```
Seamount[s_, w_, {λ_, ϕ_}][l_, p_] := s Exp[-w ((l - λ)^2 + (p - ϕ)^2)];
FaultDisplacement[p1_, p2_][p_?(VectorQ[#, NumberQ] &)] :=
 If[p1 == p2, Norm[p - p1],
  Module[{pp1, pp2, p12},
   pp1 = p - p1;
   pp2 = p - p2;
   p12 = (p1 - p2) / Norm[p1 - p2];
   If[Sign[p12.pp1] != Sign[p12.pp2],
    Norm[pp1 - pp1.p12 p12],
    Min[Norm[pp1], Norm[pp2]]]]];
```

The Ocean Floor

```
λmin = -15 °; λmax = 15 °; λe = 0; λi = 2.5 °;
ϕmin = -35 °; ϕmax = 5 °; ϕ1 = -10 °; ϕ2 = -20 °; ϕi = -25 °;
nxy = 50;
b[λ_, ϕ_] := -4000 + Seamount[3500, 1000, {-2.5 °, -25 °}][λ, ϕ] +
   Seamount[3000, 1000, {-5 °, -23 °}][λ, ϕ] +
   Seamount[3000, 1000, {-7.5 °, -27 °}][λ, ϕ] +
   Seamount[3000, 1000, {5 °, -24 °}][λ, ϕ] + Seamount[3000, 1000, {5 °, -21 °}][λ, ϕ] +
   Seamount[3000, 1000, {5 °, -18 °}][λ, ϕ] + Seamount[3000, 1000, {5 °, -15 °}][λ, ϕ] +
   Seamount[3000, 1000, {5 °, -12 °}][λ, ϕ] +
   Seamount[3750, 1000, {-2 °, -18 °}][λ, ϕ];
```

Solving the PDEs

```
In[•]:= T = 2 × 3600;
Block[{R = 6378000, ω = 2 π / (24 × 3600), f = 2 ω Sin[ϕ], g = 9.8, s = .95},
 sol = First[h /. NDSolve[
     {swe, h[λ, ϕ, 0] == Exp[-2000 FaultDisplacement[{λe, ϕ1}, {λe, ϕ2}][{λ, ϕ}]^2],
      u[λ, ϕ, 0] == 0, v[λ, ϕ, 0] == 0, h[λ, ϕmin, t] == h[λ, ϕmax, t],
      h[λmin, ϕ, t] == h[λmax, ϕ, t]}, h, {λ, λmin, λmax},
     {ϕ, ϕmin, ϕmax}, {t, 0, T}, DependentVariables → {h, u, v},
     Method → {"MethodOfLines", "SpatialDiscretization" → {"TensorProductGrid",
         "MaxPoints" → nxy, "MinPoints" -> nxy, "DifferenceOrder" → "Pseudospectral"}}]]]
```

General: Exp[−710.901] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−721.731] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−713.627] is too small to represent as a normalized machine number; precision may be lost.

General: Further output of General::munfl will be suppressed during this calculation.

```
Out[•]= InterpolatingFunction[ ]
```

Domain: {{−0.262, 0.262}, {−0.611, 0.0873}, {0., 7.20 × 10^3}}
Output: scalar

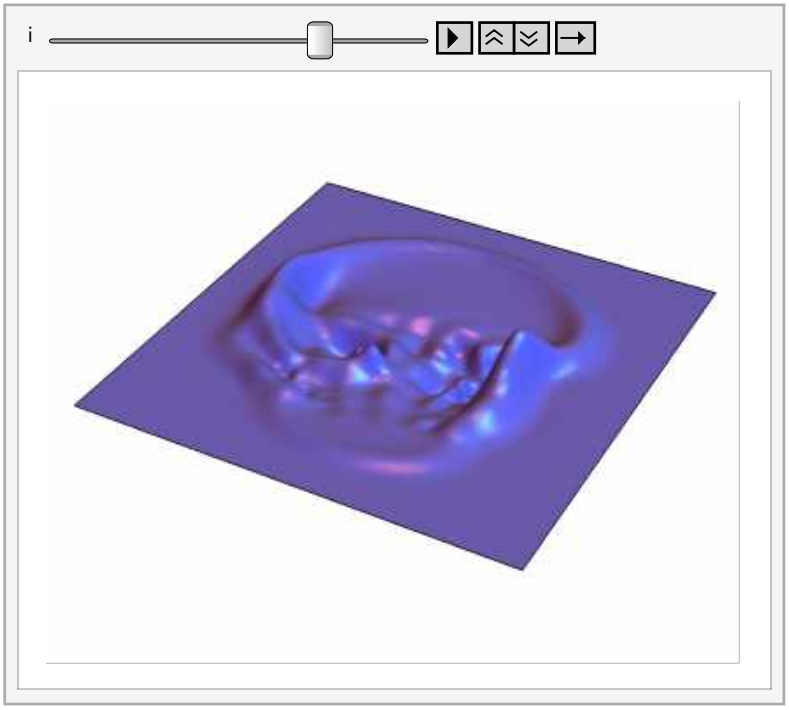
Data not in notebook; Store now »

visualize the solution

```
In[•]:= ani = Table[Plot3D[sol[λ, ϕ, t], {λ, λmin, λmax}, {ϕ, ϕmin, ϕmax},
      PlotRange → {-1, 1}, PlotPoints → 2 nxy, Mesh → None, MaxRecursion → 0,
      PlotStyle → {RGBColor[.4, .4, 1], Specularity[White, 100]},
      Boxed → False, Axes → None, Background → White], {t, 0, T, 360}];
```

In[•]:=
```
Animate[ani[[i]], {i, 1, Length[ani], 1},
 DefaultDuration → 0.25 Length[ani], SaveDefinitions → True]
```

Out[•]=



Heat equation

In[•]:=
```
sol = NDSolve[{
    ∂t u[t, x, y] == ∂x,x u[t, x, y] + ∂y,y u[t, x, y],
    u[0, x, y] == 0,
    u[t, x, 0] == Cos[t + Pi / 2], u[t, 0, y] == Sin[t],
    u[t, x, 5] == 0, u[t, 5, y] == 0}, u, {t, 0, 10}, {x, 0, 5}, {y, 0, 5}
  ]
```

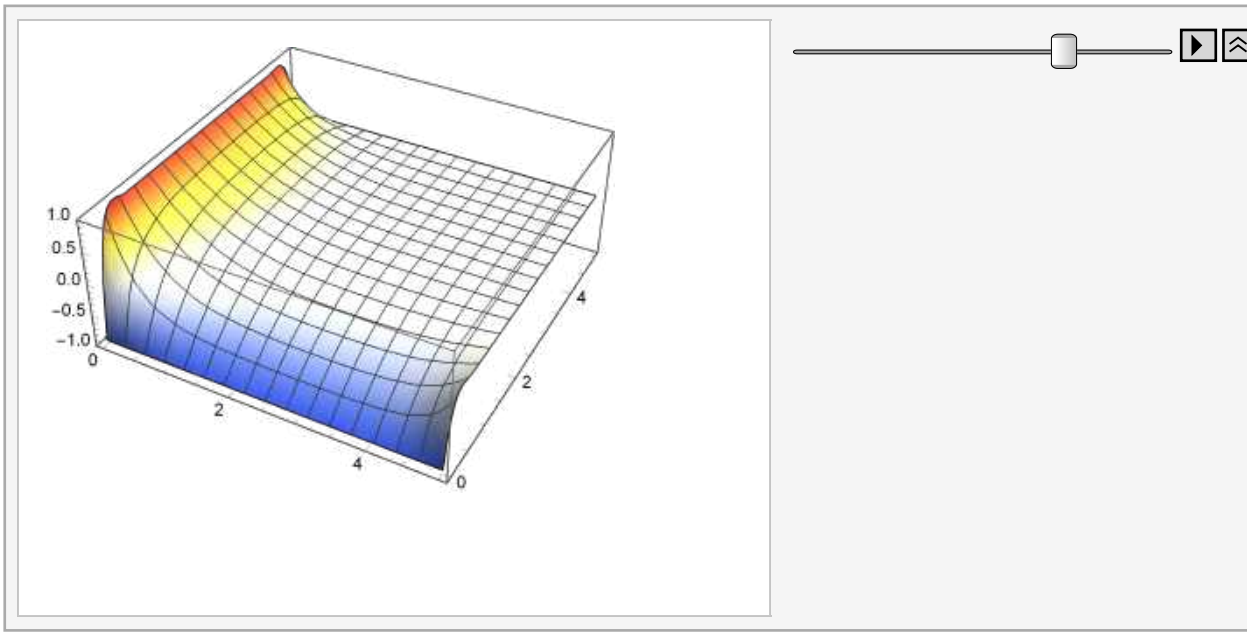
Out[•]=
```
{{u → InterpolatingFunction[ ]}}
```
Domain: {{0., 10.}, {0., 5.}, {0., 5.}}
Output: scalar

Data not in notebook; Store now »

```
Manipulate[
  Plot3D[Evaluate[u[t, x, y] /. sol], {x, 0, 5}, {y, 0, 5},
   PlotRange → {-1, 1},
   ColorFunctionScaling → False,
   ColorFunction → (ColorData["TemperatureMap"][(#3 + 1) / 2] &)
  ],
  {t, 0, 10}];
```

In[•]:=
```
Table[
  Plot3D[u[t, x, y] /. sol, {x, 0, 5}, {y, 0, 5},
   PlotRange → {-1, 1},
   ColorFunctionScaling → False,
   ColorFunction → (ColorData["TemperatureMap"][(#3 + 1) / 2] &)
  ], {t, 0, 10, 0.5}] // ListAnimate[#, SaveDefinitions → True] &
```

Out[•]=



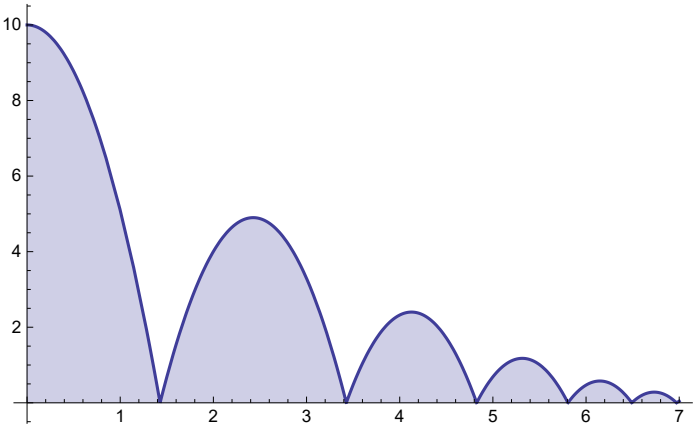
Bouncing ball ODE (WhenEvent)

In[•]:=
```
sol = NDSolve[{
    y''[t] == -9.81, y[0] == 10,
    y'[0] == 0,
    WhenEvent[y[t] == 0, y'[t] → -0.7 y'[t]]}, y, {t, 0, 7}
  ]
```

Out[•]= {{y → InterpolatingFunction[⊞ Domain: {{0., 7.}} Output: scalar]}}

In[ ]:=
```
Plot[Evaluate[y[t] /. sol[[1]]], {t, 0, 7}, PlotStyle → ColorData[1][1],
 FillingStyle → Lighter[ColorData[1][1], 0.75], PlotRange → All, Filling → Axis]
```

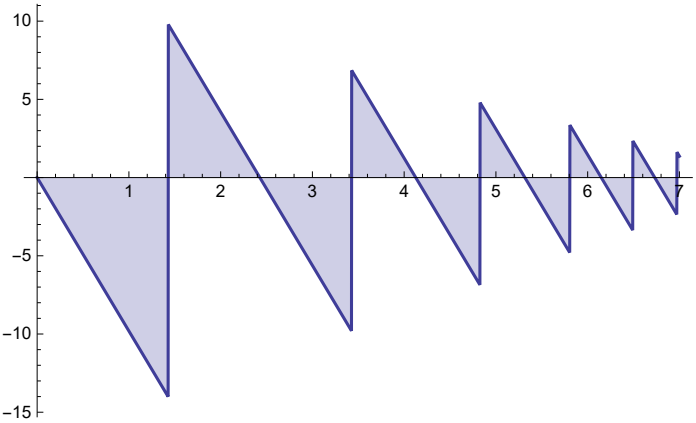
Out[ ]=




```
(*velocity profile*)
```

In[ ]:=
```
Plot[Evaluate[y'[t] /. sol[[1]]], {t, 0, 7}, PlotStyle → ColorData[1][1],
 FillingStyle → Lighter[ColorData[1][1], 0.75], PlotRange → All, Filling → Axis]
```

Out[ ]=

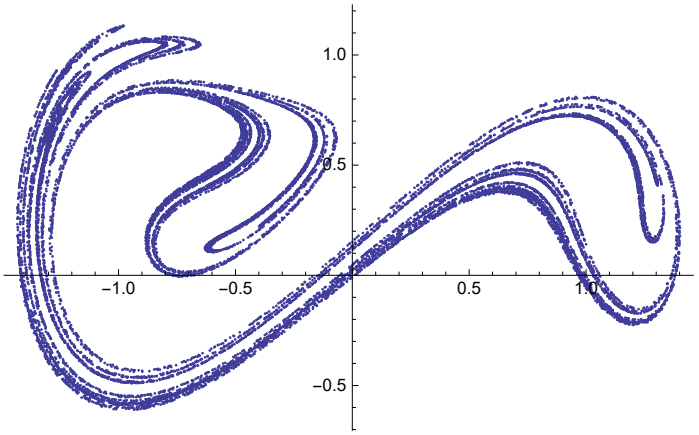


Poincare section of Duffing Equation (WhenEvent)

In[ ]:=
```
data = Reap[NDSolve[{x''[t] + 0.15 x'[t] - x[t] + x[t]^3 == 0.3 Cos[t],
     x[0] == 0, x'[0] == 0, WhenEvent[Mod[t, 2 π] == 0, Sow[{x[t], x'[t]}]]},
   x, {t, 0, 100000}, MaxSteps → ∞]][[2]];
```

In[•]:=
```
ListPlot[data, PlotStyle → Directive[ColorData[1][1], PointSize[0.0033]]]
```

Out[•]=



Vibration of a stretched string (DSolve)

In[•]:=
```
WaveEqn = ∂t,t u[t, x] == ∂x,x u[t, x];
bc = {u[t, 0] == 0, u[t, π] == 0};
ic = {u[0, x] == x^2 (π - x), u^(1,0)[0, x] == 0};
```

In[•]:=
```
dsol = DSolve[{WaveEqn, bc, ic}, u, {t, x}] /. K[1] → m;
```

In[•]:=
```
dsol
```

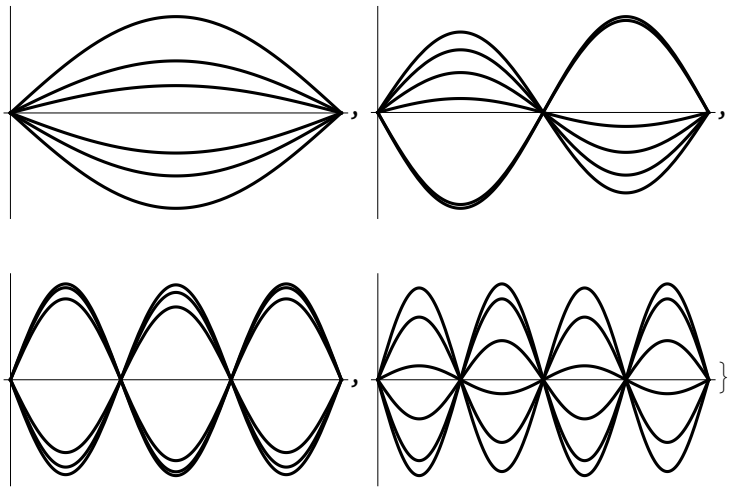
Out[•]=
$$\left\{\left\{u \to \text{Function}\left[\{t, x\}, \sum_{m=1}^{\infty} -\frac{4\left(1+2\left(-1\right)^{m}\right)\text{Cos}[t\,m]\,\text{Sin}[x\,m]}{m^3}\right]\right\}\right\}$$

In[•]:=
```
asol[t_, x_] = u[t, x] /. First[dsol] /. {∞ → 4} // Activate
```

Out[•]=
$$4\,\text{Cos}[t]\,\text{Sin}[x] - \frac{3}{2}\text{Cos}[2\,t]\,\text{Sin}[2\,x] + \frac{4}{27}\text{Cos}[3\,t]\,\text{Sin}[3\,x] - \frac{3}{16}\text{Cos}[4\,t]\,\text{Sin}[4\,x]$$

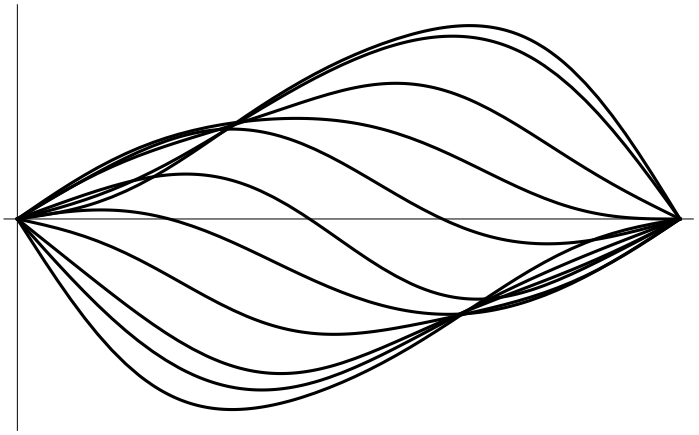
In[•]:=
```
Table[Plot[Table[asol[t, x][[m]], {t, 0, 5}] // Evaluate,
  {x, 0, π}, Ticks → False, PlotStyle → {Black}], {m, 4}]
```

Out[•]=



In[•]:=
```
Plot[Table[asol[t, x], {t, 0, 10}] // Evaluate, {x, 0, π}, Ticks → False, PlotStyle → {Black}]
```

Out[•]=



```
Animate[Plot[asol[t, x], {x, 0, π}, PlotRange → {-5, 5}, PerformanceGoal → "Speed",
    ImageSize → Medium, PlotStyle → Red], {t, 0, 10}];
```

Flow of heat in insulated bar (DSolve)

In[•]:=
```
HeatEqn = {∂t u[t, x] == ∂x,x u[t, x]};
boundaryCondition = {(∂x u[t, x] /. x → 0) == 0, (∂x u[t, x] /. x → 1) == 0};
initialCondition = u[0, x] == 100 - 80 x;
```

In[•]:=
```
sol = DSolve[{HeatEqn, boundaryCondition, initialCondition}, u[t, x], {t, x}] /. K[1] → m
```

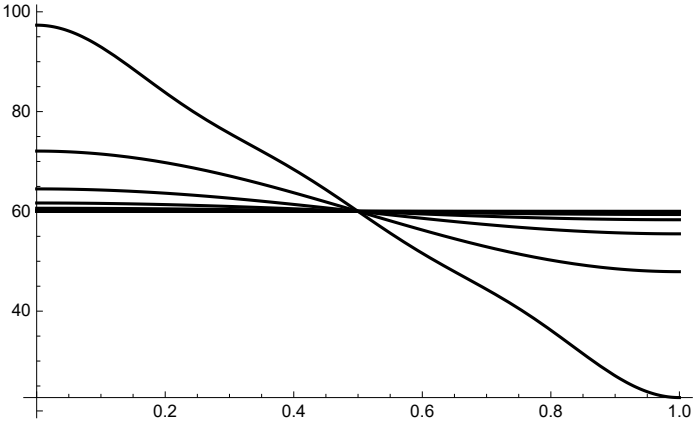
Out[•]=
$$\left\{\left\{u[t, x] \to 60 + 2 \sum_{m=1}^{\infty} -\frac{80 \left(-1 + \left(-1\right)^{m}\right) e^{-m^2 \pi^2 t} \text{Cos}[m \pi x]}{m^2 \pi^2}\right\}\right\}$$

In[•]:=
```
asol[t_, x_] = First[u[t, x] /. sol] /. {∞ → 6} // Activate // Expand
```

Out[•]=
$$60 + \frac{320\, e^{-\pi^2 t} \text{Cos}[\pi x]}{\pi^2} + \frac{320\, e^{-9 \pi^2 t} \text{Cos}[3 \pi x]}{9 \pi^2} + \frac{64\, e^{-25 \pi^2 t} \text{Cos}[5 \pi x]}{5 \pi^2}$$

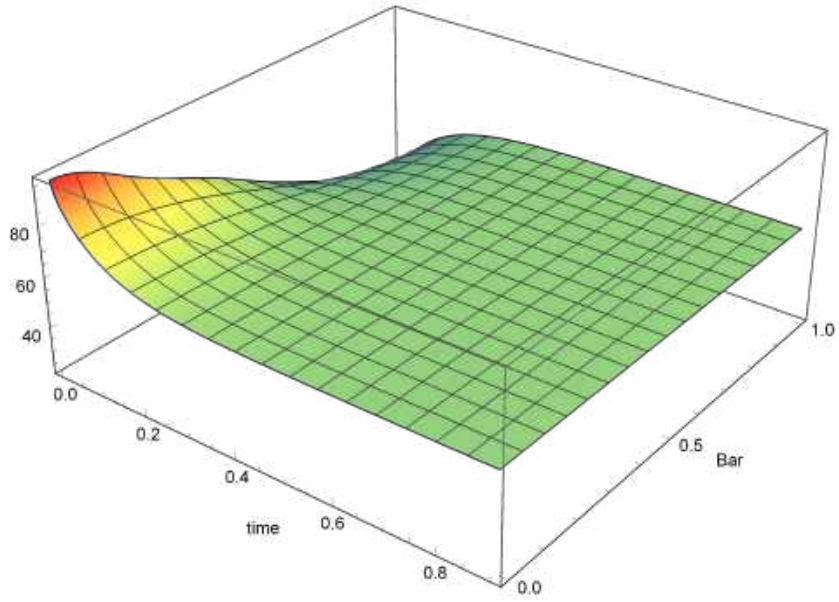
In[•]:=
```
Plot[Table[asol[t, x], {t, 0, 0.9, 0.1}] // Evaluate,
 {x, 0, 1}, PlotRange → All, PlotStyle → Black]
```

Out[•]=



In[ ]:=
```
Plot3D[asol[t, x], {t, 0, 0.9}, {x, 0, 1}, PlotPoints → 50,
 PlotRange → All, AxesLabel → {"time", "Bar"}, ColorFunction → "Rainbow"]
```

Out[ ]=



Laplacian (NDSolve)

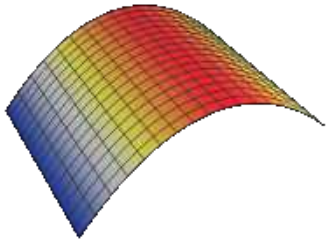
In[ ]:=
```
uif = NDSolveValue[
  {-Laplacian[u[x, y], {x, y}] == 10, u[x, 0] == 0, u[x, 1] == 1}, u, {x, 0, 1}, {y, 0, 1}]
```

Out[ ]=
```
InterpolatingFunction[
```

Domain: {{0., 1.}, {0., 1.}}
Output: scalar
]

In[ ]:=
```
Plot3D[uif[x, y], {x, 0, 1}, {y, 0, 1}, ColorFunction → "TemperatureMap",
 PlotPoints → 50, Boxed → False, Axes → False, ImageSize → Small]
```

Out[ ]=

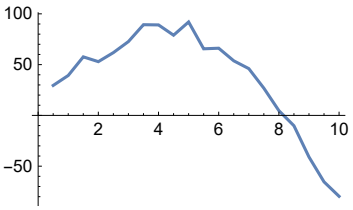


Parameter fitting to experimental data (ParametricNDSolve and FindFit)

In[ ]:=
```
data = {
    {0.5`, 29.5065424`}, {1.`, 39.3139096`},
    {1.5`, 57.6404457`}, {2.`, 52.9218489`},
    {2.5`, 61.8157312`}, {3.`, 72.70055773`},
    {3.5`, 89.3180532`}, {4.`, 89.10336725`},
    {4.5`, 78.9695762`}, {5.`, 92.00308382`},
    {5.5`, 65.5803180`}, {6.`, 66.15649415`},
    {6.5`, 53.7603732`}, {7.`, 46.07415774`},
    {7.5`, 26.8109925`}, {8.`, 4.692714582`},
    {8.5`, -10.176781`}, {9.`, -40.9309358`},
    {9.5`, -65.430306`}, {10.`, -79.6686718`}
   };
```

In[ ]:=
```
ListLinePlot[data, ImageSize → Small]
```

Out[ ]=



fit a differential equation of the form y''(t)= -w y(t), y(0)=a, y'(0)=b

In[ ]:=
```
pfun =
 y /. ParametricNDSolve[{y''[t] == -w y[t], y[0] == a, y'[0] == b}, y, {t, 0, 10}, {w, a, b}]
```

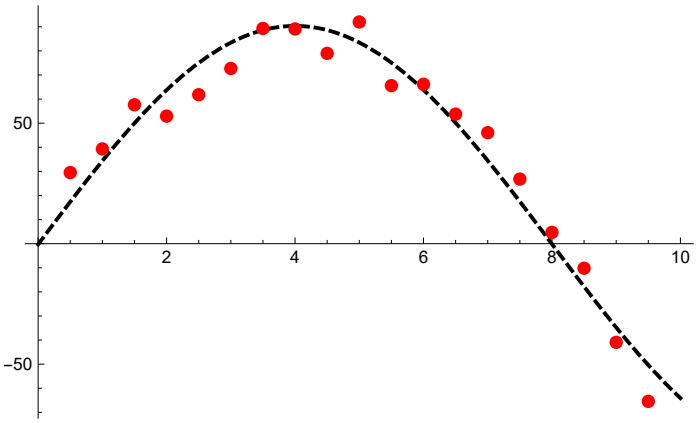
Out[ ]=
```
ParametricFunction[ Expression: y
                    Parameters: {w, a, b} ]
```

In[ ]:=
```
fit = FindFit[data, pfun[w, a, b][t], {{w, .1}, {a, -1}, {b, 1}}, t]
```

Out[ ]=
```
{w → 0.154676, a → -0.269287, b → 35.5332}
```

In[•]:=
```
Plot[pfun[Sequence @@ ({w, a, b} /. fit)][t],
 {t, 0, 10}, PlotStyle → {Black, Thick, Dashed},
 Epilog → {Red, PointSize[0.02], Point@data}, ImageSize → Medium]
```

Out[•]=




```
(* another experimental data → {time(t),temperature(u)} obtained @ position x = 1) *)
```

In[•]:=
```
data =
  {{0, 0.1848650848072292963`5.}, {0.1000000000000000056`5., 0.4030659555401104877`5.},
   {0.2000000000000000111`5., 1.0065179359796561087`5.},
   {0.3000000000000000444`5., 2.3449514014956758245`5.},
   {0.4000000000000000222`5., 3.6270776750440858471`5.},
   {0.5`5., 4.3496285197958943769`5.}, {0.6000000000000000888`5.,
    5.3844875122600814876`5.}, {0.7000000000000000666`5., 6.1749054019929401349`5.},
   {0.8000000000000000444`5., 6.9462633027730840141`5.},
   {0.9000000000000000222`5., 7.468967895745522334`5.}, {1.`5., 7.4621310261884490345`5.},
   {1.1000000000000000888`5., 8.2646419197413187874`5.},
   {1.2000000000000001776`5., 8.6446080437174099842`5.},
   {1.3000000000000000444`5., 9.0358853205431337585`5.},
   {1.4000000000000001332`5., 8.947308237961603794`5.},
   {1.5`5., 9.0867825515675981762`5.}, {1.6000000000000000888`5.,
    9.3056979339772141202`5.}, {1.7000000000000001776`5., 9.1844857316498931255`5.},
   {1.8000000000000000444`5., 9.6237838980145031798`5.},
   {1.9000000000000001332`5., 9.6456600446225024825`5.},
   {2.`5., 9.807971234911292413`5.}, {2.1000000000000000888`5., 9.6167027144315539999`5.},
   {2.2000000000000001776`5., 9.3941691794003094884`5.},
   {2.3000000000000002665`5., 9.4780092912576101583`5.},
   {2.4000000000000003553`5., 9.6427466309797189581`5.},
   {2.5`5., 10.030320328477509765`5.}, {2.6000000000000000888`5.,
    9.9281802365496094609`5.}, {2.7000000000000001776`5., 9.8560861494480640488`5.},
   {2.8000000000000002665`5., 9.5398696157849993682`5.}, {2.9000000000000003553`5.,
    9.9829742313712692692`5.}, {3.`5., 9.6074787805808767871`5.}};
```

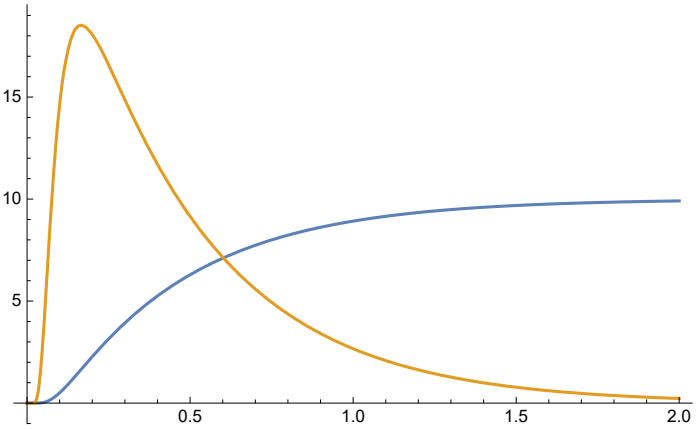
In[•]:=
```
pfun = ParametricNDSolveValue[{D[u[t, x], t] == a D[u[t, x], x, x],
   u[0, x] == 20 UnitStep[-x], u[t, 0] == 20, u[t, 2] == 0}, u, {t, 0, 3}, {x, 0, 2}, {a}]
```

Out[•]=
```
ParametricFunction[ $Failed  Expression: u
                             Parameters: {a} ]
```

```
(* first we pass value of param to pfun and then the domain values →{t,x} *)
```

In[•]:=
```
Plot[Evaluate[{pfun[a][t, 1], D[pfun[a][t, 1], t]} /. a → 1], {t, 0, 2}]
```

Out[•]=

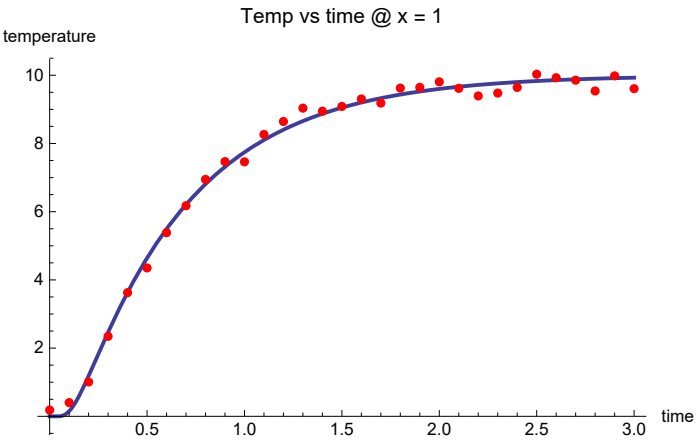


In[•]:=
```
param = FindFit[data, pfun[a][t, 1], a, t]
```

Out[•]= {a → 0.70144}

In[•]:=
```
Plot[pfun[a /. param][t, 1], {t, 0, 3}, PlotStyle → {Thick, ColorData[1][1]},
 Epilog → {Red, PointSize[0.015], Point@data},
 AxesLabel → {"time", "temperature"}, PlotLabel → "Temp vs time @ x = 1"]
```

Out[•]=



three modern PDEs

Shock Waves and Burger's Equation (DSolve)

In[•]:=
```
BurgersEqn = D[u[t, x], t] + u[t, x] × D[u[t, x], x] == ϵ D[u[t, x], x, x];
```

In[•]:=
```
initialCondition = u[0, x] == Piecewise[{{1, x < 0}}];
```

In[•]:=
```
dsol = DSolveValue[{BurgersEqn, initialCondition}, u[t, x], {t, x}]
```

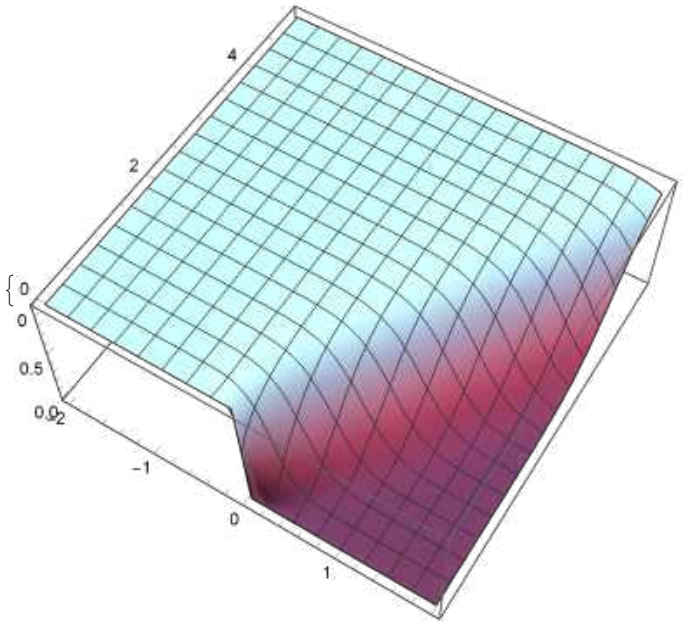
Out[•]=
$$\frac{1}{1+\frac{e^{-\frac{t-2x}{4\epsilon}}\left(1+\text{Erf}\left[\frac{x}{2\sqrt{t\epsilon}}\right]\right)}{1+\text{Erf}\left[\frac{t-x}{2\sqrt{t\epsilon}}\right]}}$$

In[•]:=
```
Table[Plot3D[Evaluate[dsol /. ϵ → j], {x, -2, 2},
  {t, 0.001, 5}, ColorFunction → "CandyColors", PlotPoints → 25,
  ImageSize → Medium], {j, {1/10, 1/100, 1/10000}}]
```

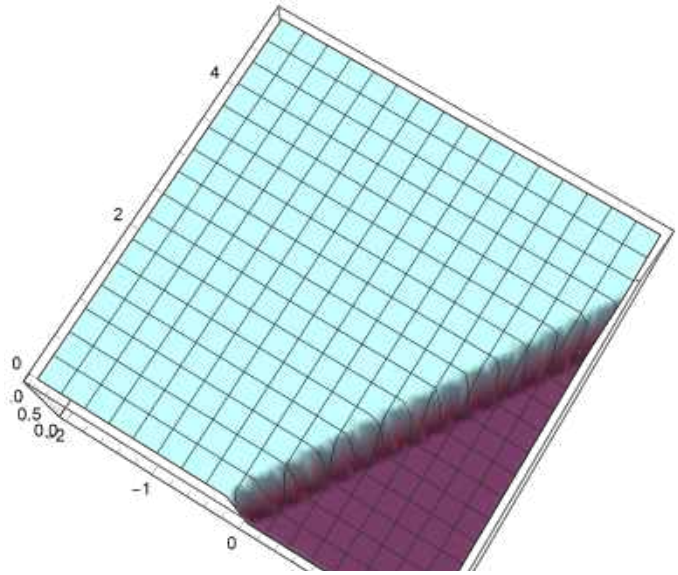
General: Exp[−10002.2] is too small to represent as a normalized machine number; precision may be lost.

General: Exp[−9689.9] is too small to represent as a normalized machine number; precision may be lost.

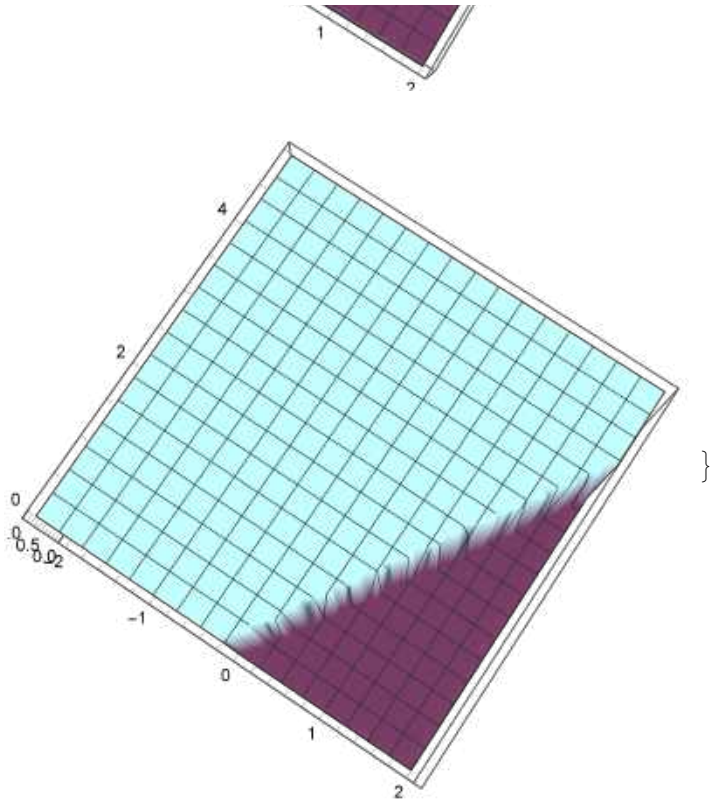
Out[•]=



,



,



}

Black Scholes model (DSolve)

In[•]:=
```
BlackScholesModel = {D[c[t, s], t] + 1/2 s^2 σ^2 D[c[t, s], s, s] + r s D[c[t, s], s] - r c[t, s] == 0,
    c[T, s] == Max[s - k, 0]};
```

In[•]:=
```
dsol = DSolveValue[BlackScholesModel, c[t, s], {t, s}]
```

Out[•]=
$$\frac{1}{2} e^{-r T} \left(-e^{r t} k \operatorname{Erfc}\left[\frac{(t-T)\left(2r-\sigma^2\right) + 2\operatorname{Log}[k] - 2\operatorname{Log}[s]}{2\sqrt{2}\sqrt{-t+T}\,\sigma} \right] + e^{r T} s \operatorname{Erfc}\left[\frac{(t-T)\left(2r+\sigma^2\right) + 2\operatorname{Log}[k] - 2\operatorname{Log}[s]}{2\sqrt{2}\sqrt{-t+T}\,\sigma} \right] \right)$$

In[•]:=
```
dsol /. {t → 0, s → 100, k → 100, σ → 0.2, T → 1, r → 0.05}
```

Out[•]=
```
10.4506
```

In[•]:=
```
FinancialDerivative[{"European", "Call"}, {"StrikePrice" → 100.00, "Expiration" → 1},
 {"InterestRate" → 0.05, "Volatility" → 0.2, "CurrentPrice" → 100}]
```

Out[•]=
```
10.4506
```

Quantum particle in a box (DSolve)

```
In[•]:= SchrodingerEquation = I ħ ∂t ψ[t, x] == -ħ^2/(2 m) ∂x,x ψ[t, x];

In[•]:= BC = {ψ[t, 0] == 0, ψ[t, d] == 0};

In[•]:= IC = ψ[0, x] == 1/√d Sin[π x/d] + 1/√d Sin[2 π x/d];

In[•]:= sol = DSolve[{SchrodingerEquation, BC, IC}, ψ[t, x], {t, x}]
```

$$\text{Out[•]}= \left\{\left\{\psi[t, x] \to \frac{e^{-\frac{2 i \pi^2 t \hbar}{d^2 m}} \left(e^{\frac{3 i \pi^2 t \hbar}{2 d^2 m}} + 2 \cos\left[\frac{\pi x}{d}\right]\right) \sin\left[\frac{\pi x}{d}\right]}{\sqrt{d}}\right\}\right\}$$

```
(* computing the probability density *)

In[•]:= ρ[t_, x_] = Simplify[ComplexExpand[Conjugate[ψ[t, x]] ψ[t, x] /. First[sol]], d > 0]
```

$$\text{Out[•]}= \frac{\left(3 + 2 \cos\left[\frac{2 \pi x}{d}\right] + 2 \cos\left[\frac{\pi (2 d m x - 3 \pi t \hbar)}{2 d^2 m}\right] + 2 \cos\left[\frac{\pi (2 d m x + 3 \pi t \hbar)}{2 d^2 m}\right]\right) \sin\left[\frac{\pi x}{d}\right]^2}{d}$$

```
In[•]:= Clear[t, x];

In[•]:= ρe[t_, x_] = ρ[t, x] /. {ħ → 1.055*^-34, m → 9.109*^-31, d → 10^-9}
```

$$\text{Out[•]}= 1\,000\,000\,000 \left(3 + 2 \cos\left[1.72444 \times 10^{48} \left(-9.94314 \times 10^{-34} t + 1.8218 \times 10^{-39} x\right)\right] + 2 \cos\left[1.72444 \times 10^{48} \left(9.94314 \times 10^{-34} t + 1.8218 \times 10^{-39} x\right)\right] + 2 \cos[2\,000\,000\,000\,\pi x]\right) \sin[1\,000\,000\,000\,\pi x]^2$$

```
In[•]:= Animate[Plot[ρe[t, x], {x, 0, 10^-9}, PlotRange → {0, 4*^9},
  PlotStyle → ColorData[1][1], ImageSize → Medium, Filling → Bottom,
  AxesLabel → {"x(m)", "ρe (m^-1)"}], {t, 0, 3.66*^-15}]
```

solving PDEs over Regions

Dirichlet problem in a rectangle (DSolve)

```
In[•]:= LapEqn = ∂x,x u[x, y] + ∂y,y u[x, y] == 0; (* ∇^2_{x,y} u[x,y]==0 *)
BC = DirichletCondition[
   u[x, y] == Piecewise[{{UnitTriangle[2 x - 1], y == 0 || y == 2}}, 0], True];
Ω = Rectangle[{0, 0}, {1, 2}];
```

In[•]:=
```
sol = FullSimplify[DSolve[{LapEqn, BC}, u[x, y], {x, y} ∈ Ω]]
```

Out[•]=
$$\left\{\left\{u[x, y] \to \sum_{K[1]=1}^{\infty} \frac{8 \text{Cosh}\left[\pi \left(-1+y\right) K[1]\right] \text{Sech}[\pi K[1]] \text{Sin}\left[\frac{1}{2} \pi K[1]\right] \text{Sin}[\pi x K[1]]}{\pi^2 K[1]^2}\right\}\right\}$$

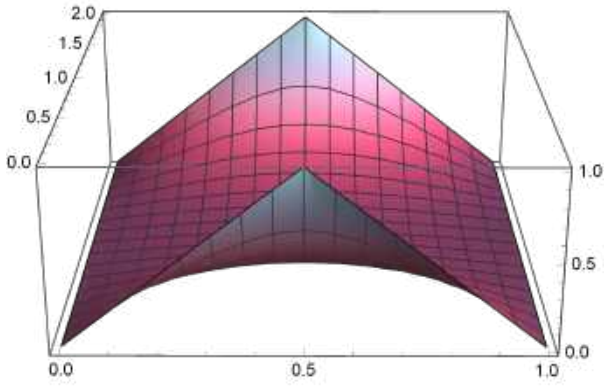
In[•]:=
```
asol = u[x, y] /. sol[[1]] /. {∞ → 300} // Activate;
```

In[•]:=
```
Plot3D[asol, {x, 0, 1}, {y, 0, 2}, PlotRange → All, ColorFunction → "CandyColors"]
```

... General: Sech[713.142] is too small to represent as a normalized machine number; precision may be lost.

... General: Sech[719.425] is too small to represent as a normalized machine number; precision may be lost.

Out[•]=



Poisson type equations (NDSolve)

In[•]:=
```
Ω = Disk[];
op = -Laplacian[u[x, y], {x, y}] == 1;
```

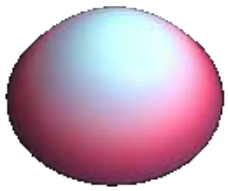
In[•]:=
```
Γ_D = DirichletCondition[u[x, y] == 0, x^2 + y^2 ≥ 0.5];
```

In[•]:=
```
uif = First@NDSolve[{op, Γ_D}, u, {x, y} ∈ Ω]
```

Out[•]=
```
{u → InterpolatingFunction[ ⊞ 〰 Domain: {{-1., 1.}, {-1., 1.}}
                            Output: scalar ]}
```

In[•]:=
```
Plot3D[Evaluate[u[x, y] /. uif], {x, y} ∈ Ω, ColorFunction → "CandyColors",
 Mesh → None, PlotPoints → 50, Boxed → False, Axes → False, ImageSize → Small]
```

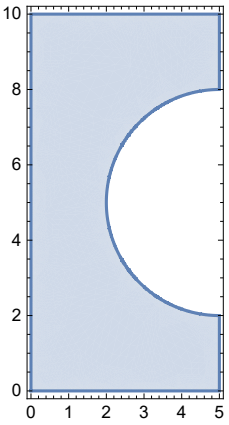
Out[•]=



In[•]:=
```
Ω = ImplicitRegion[0 ≤ x ≤ 5 && 0 ≤ y ≤ 10 && ! ((x - 5)^2 + (y - 5)^2 ≤ 3^2), {x, y}];
```

In[•]:=
```
RegionPlot[Ω, AspectRatio → {1, 1}, ImageSize → Small]
```

Out[•]=



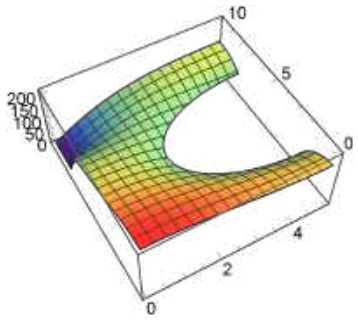
In[•]:=
```
op = -Laplacian[u[x, y], {x, y}] - 20;
```

In[•]:=
```
Γ_D = {DirichletCondition[u[x, y] == 0, x == 0 && 8 ≤ y ≤ 10],
    DirichletCondition[u[x, y] == 100, ((x - 5)^2 + (y - 5)^2 == 3^2)]};
```

In[•]:=
```
uif = NDSolveValue[{op == 0, Γ_D}, u, {x, y} ∈ Ω];
```

In[•]:=
```
Plot3D[uif[x, y], {x, y} ∈ Ω, ColorFunction → "Rainbow", ImageSize → Small]
```

Out[•]=



In[•]:=
```
Γ_D = DirichletCondition[u[x, y] == 0, x == 0 && 8 ≤ y ≤ 10];
```

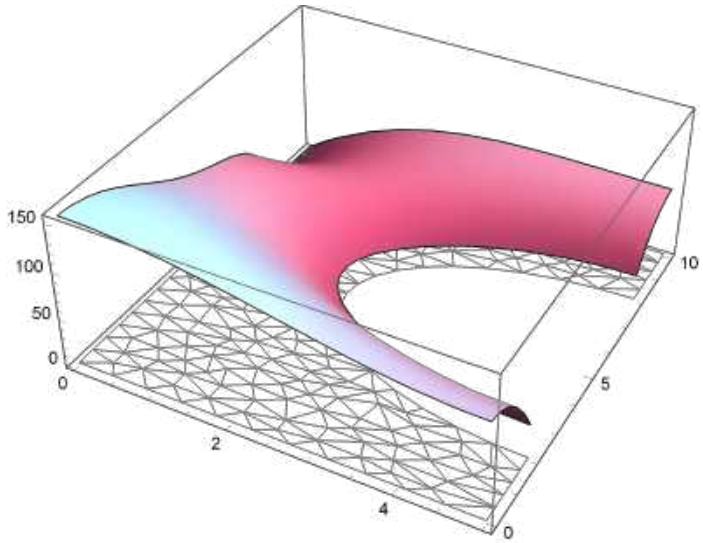
In[•]:=
```
Γ_N = NeumannValue[-2 u[x, y], ((x - 5)^2 + (y - 5)^2 == 3^2)];
```

In[•]:=
```
uif = NDSolveValue[{op == Γ_N, Γ_D}, u, {x, y} ∈ Ω];
```

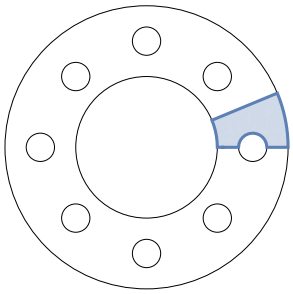
In[•]:=
```
Needs["NDSolve`FEM`"]
```

```
Show[
 Plot3D[uif[x, y], {x, y} ∈ uif["ElementMesh"],
  Mesh → None, ColorFunction → "CandyColors"],
 Graphics3D[{Opacity[0.2], EdgeForm[Gray], FaceForm[None],
   NDSolve`FEM`ElementMeshToGraphicsComplex[uif["ElementMesh"], "CoordinateConversion" ->
     (Join[#, ConstantArray[{0.}, {Length[#]}], 2] &)]}], ImageSize → Medium]
```

Out[ ]=



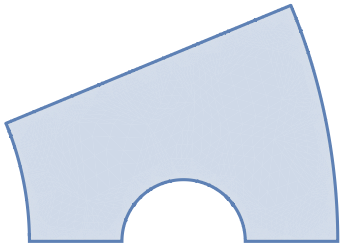
some transient problem (NDSolve)



In[ ]:=

```
RegionPlot[
 Ω = ImplicitRegion[y ≥ 0 && (x - 3/2)^2 + y^2 ≥ 1/25 && x^2 + y^2 ≥ 1 && x^2 + y^2 ≤ 4 && y ≤ x Tan[π/8], {x, y}],
 AspectRatio → {1, 1}, Frame → False, ImageSize → Small]
```

Out[ ]=



In[ ]:=

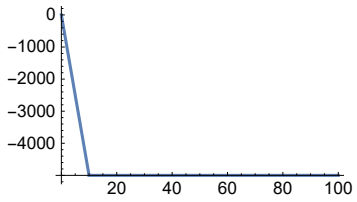
```
Clear[eqn, Γ];
```

In[]:=
```
eqn[t_, x_, y_] = -10 Laplacian[u[t, x, y], {x, y}];
```

In[]:=
```
Γ_Dc[t_] = {DirichletCondition[u[t, x, y] == 200, x^2 + y^2 == 1],
    DirichletCondition[u[t, x, y] == 10., x^2 + y^2 == 4]};
```

In[]:=
```
Plot[Piecewise[{{-500. t, t ≤ 10}, {-5000, t > 10}}], {t, 0, 100}, ImageSize → Small]
```

Out[]=

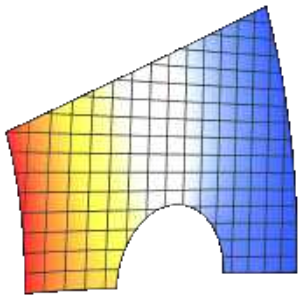


In[]:=
```
Γ_Nc[t_] = NeumannValue[If[t ≤ 10, -500. t, -5000], (x - 3/2)^2 + y^2 == 1/25];
```

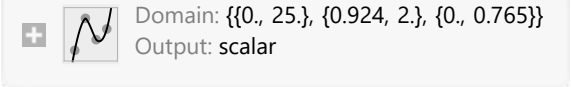
In[]:=
```
unit = NDSolveValue[{eqn[0, x, y] == 0, Γ_Dc[0]}, u[0, x, y], {x, y} ∈ Ω];
```

In[]:=
```
Plot3D[unit, {x, y} ∈ Ω, ColorFunction → "TemperatureMap",
 ViewPoint → Above, Boxed → False, Axes → False, ImageSize → Small]
```

Out[]=



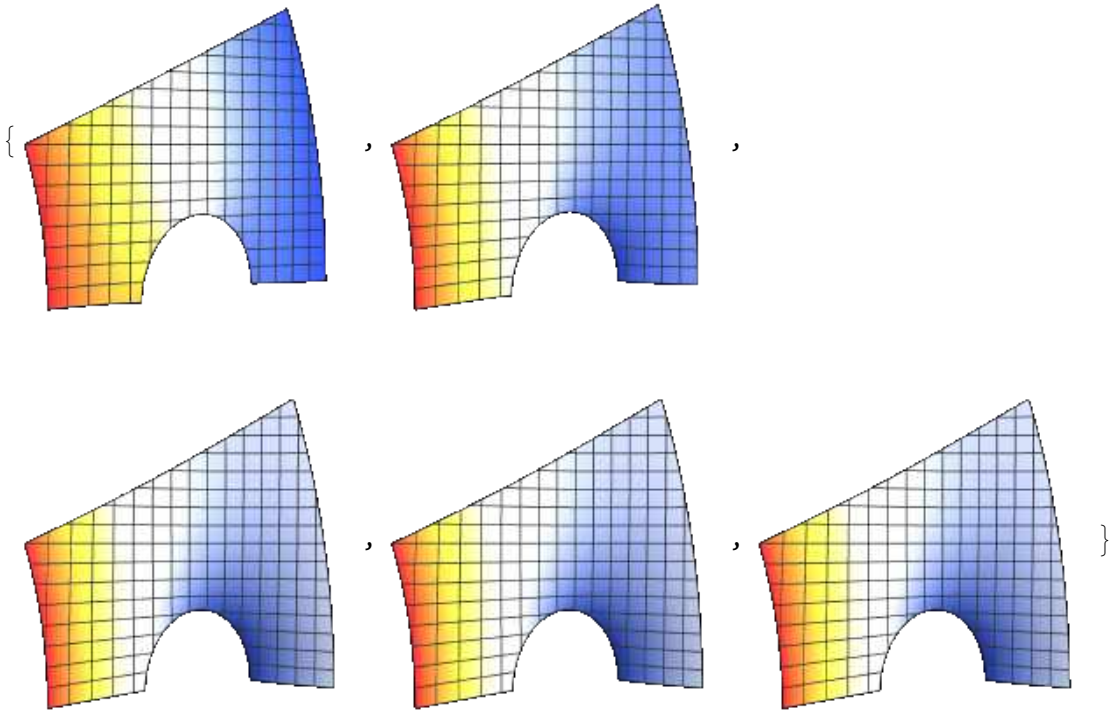
In[]:=
```
uif = NDSolveValue[{D[u[t, x, y], t] + eqn[t, x, y] == Γ_Nc[t], Γ_Dc[0], u[0, x, y] == unit},
  u, {t, 0, 25}, {x, y} ∈ Ω]
```

Out[]=
```
InterpolatingFunction[
```


```
]
```

In[•]:=
```
Table[Plot3D[uif[t, x, y], {x, y} ∈ Ω, ColorFunction → "TemperatureMap", ViewPoint → Above,
  Boxed → False, Axes → False, ImageSize → Small], {t, {0, 5, 10, 15, 25}}]
```

Out[•]=



system of PDEs over a region

Stokes flow in a duct (NDSolve)

In[232]:=
```
ClearAll["Global`*"]
```

```
(*
{∇{x,y} · ({{-μ,0},{0,-μ}}.∇{x,y}u[x,y]) +p^(1,0)[x,y],
   ∇{x,y} · ({{-μ,0},{0,-μ}}.∇{x,y}v[x,y]) +p^(0,1)[x,y],u^(1,0)[x,y]+v^(0,1)[x,y]}=={0,0,0}/.μ→1
*)
```

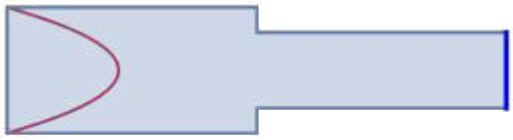
In[280]:=
```
ClearAll[iStokesFlow, StokesFlow];
iStokesFlow[a_, depVars : {u_, v_, ___, p_},
   X : {x_, y_, ___}, msghead_Symbol] := Module[{diag},
   diag = -a * IdentityMatrix[Length[depVars] - 1];
   Join[(Inactive[Div][diag.Inactive[Grad][#, X], X] & /@ Most[depVars]) +
      (D[p, #] & /@ X), {Total[MapThread[D[#1, #2] &, {Most[depVars], X}]]}]
  ];
```

In[305]:=
```
PiecewiseExpand[Piecewise[{{mu, True}}]]
```

Out[305]=
```
mu
```

In[282]:=
```
StokesFlow[mu_, depVars : {u_, v_, ___, p_}, X : {x_, y_, ___}] /;
   Length[X] ≥ 2 && Length[depVars] == Length[X] + 1 :=
  With[{res = Catch[iStokesFlow[PiecewiseExpand[Piecewise[{{mu, True}}]], depVars, X,
       StokesFlow], NDSolve`FEM`FEMError[___]]}, res /; Unevaluated[res] =!= $Failed
  ];
```

In[236]:=
```
Ω = ImplicitRegion[0 ≤ x ≤ 2 && 0 ≤ y ≤ 0.5 && ! (x ≥ 1 && y ≤ 0.1) && ! (x ≥ 1 && y ≥ 0.4), {x, y}];
```



In[132]:=
```
(* programatically generate the PDEs *)
```

In[283]:=
```
pde = StokesFlow[1, {u[x, y], v[x, y], p[x, y]}, {x, y}] == {0, 0, 0}
```

Out[283]=
```
{∇{x,y} · ({{-1, 0}, {0, -1}}.∇{x,y} u[x, y]) + p^(1,0)[x, y],
  ∇{x,y} · ({{-1, 0}, {0, -1}}.∇{x,y} v[x, y]) + p^(0,1)[x, y],
  v^(0,1)[x, y] + u^(1,0)[x, y]} == {0, 0, 0}
```

In[135]:=
```
(* manually enter the PDEs *)
```

In[287]:=
```
pdes = {Inactive[Div][
       {{-μ, 0}, {0, -μ}}.Inactive[Grad][u[x, y], {x, y}], {x, y}] + D[p[x, y], x],
    Inactive[Div][{{-μ, 0}, {0, -μ}}.Inactive[Grad][v[x, y], {x, y}], {x, y}] +
     D[p[x, y], y],
    D[u[x, y], x] + D[v[x, y], y]} == {0, 0, 0} /. μ → 1
```

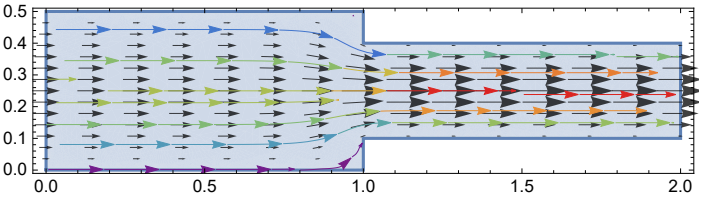
Out[287]=
```
{∇{x,y} · ({{-1, 0}, {0, -1}}.∇{x,y} u[x, y]) + p^(1,0)[x, y],
  ∇{x,y} · ({{-1, 0}, {0, -1}}.∇{x,y} v[x, y]) + p^(0,1)[x, y],
  v^(0,1)[x, y] + u^(1,0)[x, y]} == {0, 0, 0}
```

In[288]:=
```
ΓD = {
   DirichletCondition[u[x, y] == 4 * 0.3 * y * (0.5 - y) / (0.41)^2, x == 0.],
   DirichletCondition[{u[x, y] == 0., v[x, y] == 0.}, 0. < x < 2],
   DirichletCondition[{p[x, y] == 0.}, x == 2]
  };
```

In[289]:=
```
{uvel, vvel, pressure} = NDSolveValue[{pdes, ΓD}, {u, v, p}, {x, y} ∈ Ω,
   Method → {"FiniteElement", "InterpolationOrder" → {u → 2, v → 2, p → 1}}];
```

In[290]:=
```
Show[RegionPlot[Ω, AspectRatio → {1, 1}],
 VectorPlot[{uvel[x, y], vvel[x, y]}, {x, y} ∈ Ω,
  VectorStyle → {Opacity[0.75], Black}, StreamPoints → 12,
  StreamColorFunction → "Rainbow", StreamColorFunctionScaling → True]
]
```

Out[290]=



Structural mechanics (NDSolve)

In[356]:=
```
gr  =                                   ;
```

In[357]:=
```
Needs["NDSolve`FEM`"]
```

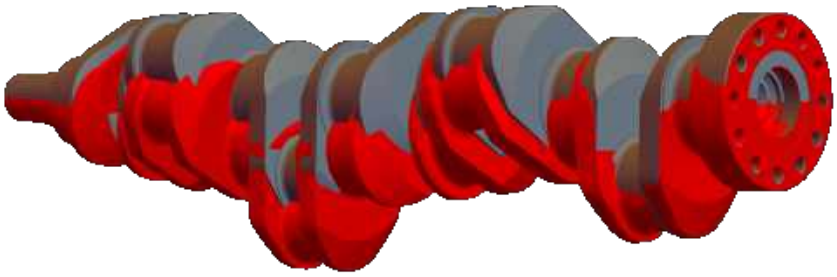
In[358]:=
```
mesh = ToElementMesh[DiscretizeGraphics@gr, "MeshOrder" → 1]
```

Out[358]=
```
ElementMesh[{{-93.9459, 93.9356}, {0., 890.1}, {-102.45, 73.6294}},
 {TetrahedronElement[<175 355>]}]
```

In[359]:=
```
stressOperator[Y_, ν_] :=
  {Inactive[Div][({{{0, 0, -((Y ν)/((1 - 2 ν) (1 + ν)))}, {0, 0, 0}, {-(Y/(2 (1 + ν))),
         0, 0}}.Inactive[Grad][w[x, y, z], {x, y, z}]), {x, y, z}] +
   Inactive[Div][({{{0, -((Y ν)/((1 - 2 ν) (1 + ν))), 0}, {-(Y/(2 (1 + ν))), 0, 0},
       {0, 0, 0}}.Inactive[Grad][v[x, y, z], {x, y, z}]), {x, y, z}] +
   Inactive[Div][({{{-((Y (1 - ν))/((1 - 2 ν) (1 + ν))), 0, 0}, {0, -(Y/(2 (1 + ν))), 0},
       {0, 0, -(Y/(2 (1 + ν)))}}.Inactive[Grad][u[x, y, z], {x, y, z}]), {x, y, z}],
   Inactive[Div][({{0, 0, 0}, {0, 0, -((Y ν)/((1 - 2 ν) (1 + ν)))},
       {0, -(Y/(2 (1 + ν))), 0}}.Inactive[Grad][w[x, y, z], {x, y, z}]), {x, y, z}] +
   Inactive[Div][({{{0, -(Y/(2 (1 + ν))), 0}, {-((Y ν)/((1 - 2 ν) (1 + ν))), 0, 0},
       {0, 0, 0}}.Inactive[Grad][u[x, y, z], {x, y, z}]), {x, y, z}] +
   Inactive[Div][({{{-(Y/(2 (1 + ν))), 0, 0}, {0, -((Y (1 - ν))/((1 - 2 ν) (1 + ν))), 0},
       {0, 0, -(Y/(2 (1 + ν)))}}.Inactive[Grad][v[x, y, z], {x, y, z}]), {x, y, z}],
   Inactive[Div][({{0, 0, 0}, {0, 0, -(Y/(2 (1 + ν)))}, {0, -((Y ν)/((1 - 2 ν) (1 + ν))),
         0}}.Inactive[Grad][v[x, y, z], {x, y, z}]), {x, y, z}] + Inactive[Div][
     ({{0, 0, -(Y/(2 (1 + ν)))}, {0, 0, 0}, {-((Y ν)/((1 - 2 ν) (1 + ν))), 0, 0}}.
       Inactive[Grad][u[x, y, z], {x, y, z}]), {x, y, z}] +
   Inactive[Div][({{{-(Y/(2 (1 + ν))), 0, 0}, {0, -(Y/(2 (1 + ν))), 0},
       {0, 0, -((Y (1 - ν))/((1 - 2 ν) (1 + ν)))}}.
      Inactive[Grad][w[x, y, z], {x, y, z}]), {x, y, z}]};

(*specify constraints such that the boundary
 of the crankshaft is held fixed at these positions *)

In[378]:= Γ_D =
  DirichletCondition[{u[x, y, z] == 0., v[x, y, z] == 0., w[x, y, z] == 0.}, {y ≤ 0.1, y ≥ 890}];

(*specify boundary load pushing in the negative z direction*)

In[379]:= Γ_n = NeumannValue[-10000.,
    ((217 ≤ y ≤ 256) && (40 ≤ z ≤ 103)) || ((584 ≤ y ≤ 622) && (40 ≤ z ≤ 103))];

(*solving the coupled PDEs*)

In[384]:= AbsoluteTiming[{uif, vif, wif} = NDSolveValue[
    {stressOperator[10^9, 33/100] == {0, 0, Γ_n}, Γ_D}, {u, v, w}, {x, y, z} ∈ mesh];]

Out[384]= {6.33607, Null}
```

In[385]:=
```
Show[
 ToBoundaryMesh[mesh][
  "Wireframe"["MeshElementStyle" → {Directive[EdgeForm[], FaceForm[Gray]]}]],
 ElementMeshDeformation[mesh, {uif, vif, wif}, "ScalingFactor" → 100][
  "Wireframe"[AspectRatio → Automatic,
   "MeshElementStyle" → Directive[{EdgeForm[], FaceForm[Red]}]]],
 ImageSize → 500, ViewPoint → {-2, -2, 0}
]
```

Out[385]=



symbolic Eigenfunction Problems

Basic problem (DSolve)

In[•]:=
```
sol = DSolve[{y''[x] + λ y[x] == 0, y[0] == 0, y[π] == 0}, y[x], x]
```

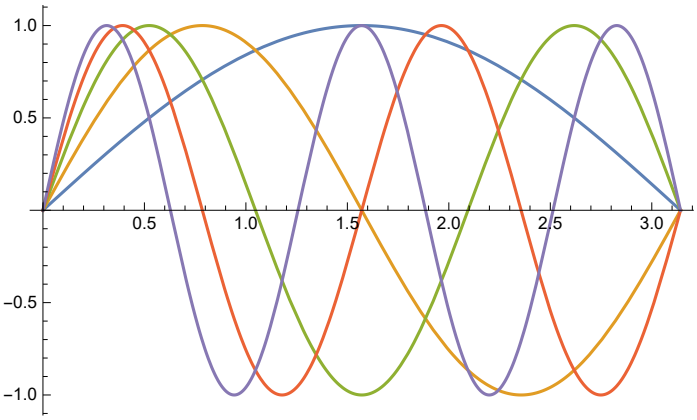
Out[•]=
$$\left\{\left\{y[x] \to \begin{cases} c_1 \text{Sin}\left[x \sqrt{\lambda}\right] & n \in \mathbb{Z} \&\& n \geq 1 \&\& \lambda == n^2 \\ 0 & \text{True} \end{cases}\right\}\right\}$$

In[•]:=
```
eigfun = Table[y[x] /. sol[[1]] //. {n → i, λ → n^2} /. C[1] → 1, {i, 5}]
```

Out[•]=
```
{Sin[x], Sin[2 x], Sin[3 x], Sin[4 x], Sin[5 x]}
```

In[•]:=
```
Plot[Evaluate[eigfun], {x, 0, π}]
```

Out[•]=



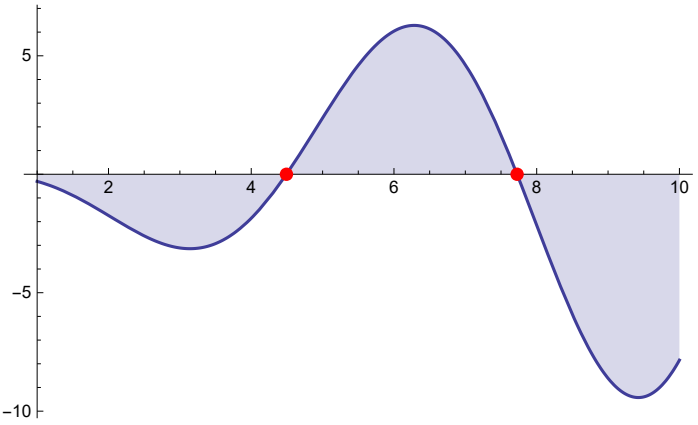
Robin problem for an ODE (DSolve)

```
In[•]:= sol = DSolve[{y''[x] + λ^2 y[x] == 0, y[0] + y'[0] == 0, y[1] == 0}, y[x], x]
```

$$\text{Out[•]= } \left\{\left\{y[x] \to \begin{cases} c_1 \left(-\lambda \text{Cos}[x\lambda] + \text{Sin}[x\lambda]\right) & \lambda \text{Cos}[\lambda] == \text{Sin}[\lambda] \\ 0 & \text{True} \end{cases}\right\}\right\}$$

```
In[•]:= roots = λ /. Solve[λ Cos[λ] == Sin[λ] && 1 < λ < 10];
        data = Table[{λ, 0}, {λ, roots}];

In[•]:= Plot[λ Cos[λ] - Sin[λ], {λ, 1, 10}, Filling → Axis,
         PlotStyle → ColorData[1][1], Epilog → {Red, PointSize[0.02], Point@data}]
```

Out[•]=




```
In[•]:= FindRoot[λ Cos[λ] - Sin[λ] == 0, {λ, #}] & /@ {4, 7}
```

$$\text{Out[•]= } \{\{\lambda \to 4.49341\}, \{\lambda \to 7.72525\}\}$$

```
        (* we could also have added assumption to DSolve *)

In[•]:= dsol =
         DSolve[{y''[x] + λ^2 y[x] == 0, y[0] + y'[0] == 0, y[1] == 0}, y[x], x, Assumptions → 1 < λ < 10]
```

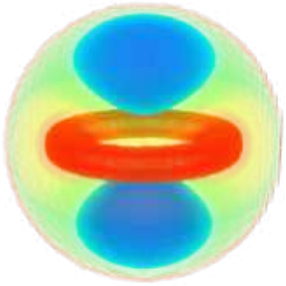
$$\text{Out[•]= } \left\{\left\{y[x] \to \begin{cases} c_1 \left(-\lambda \text{Cos}[x\lambda] + \text{Sin}[x\lambda]\right) & \lambda == 4.49\ldots \;||\; \lambda == 7.73\ldots \\ 0 & \text{True} \end{cases}\right\}\right\}$$

Eigenfunction of the Laplacian in a Ball (DEigensystem)

```
In[•]:= L = -Laplacian[u[x, y, z], {x, y, z}];

In[•]:= B = DirichletCondition[u[x, y, z] == 0, True];

In[•]:= {vals, funs} = DEigensystem[{L, B}, u[x, y, z], {x, y, z} ∈ Ball[{0, 0, 0}, 2], 7];
```

In[•]:=
```
DensityPlot3D[funs[[7]] // N // Evaluate, {x, y, z} ∈ Ball[{0, 0, 0}, 2],
  Boxed → False, Axes → False, ColorFunction → Hue, ImageSize → Small]
```

Out[•]=



NDEigensystem

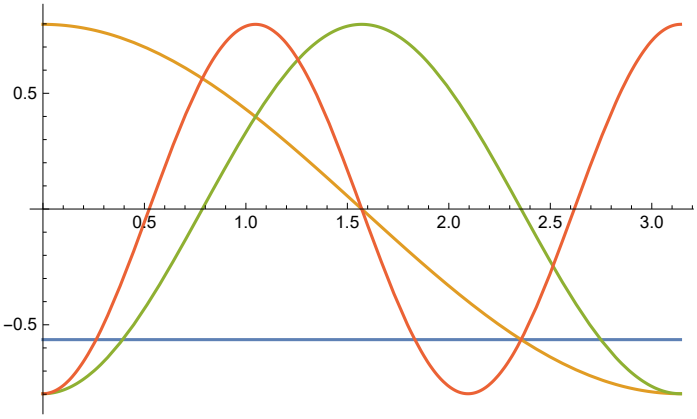
eigen modes

$$-\nabla_x^2 u = \lambda u$$

In[•]:=
```
{vals, funs} = NDEigensystem[-Laplacian[u[x], {x}], u[x], {x, 0, Pi}, 4];
```

In[•]:=
```
Plot[funs, {x, 0, Pi}]
```

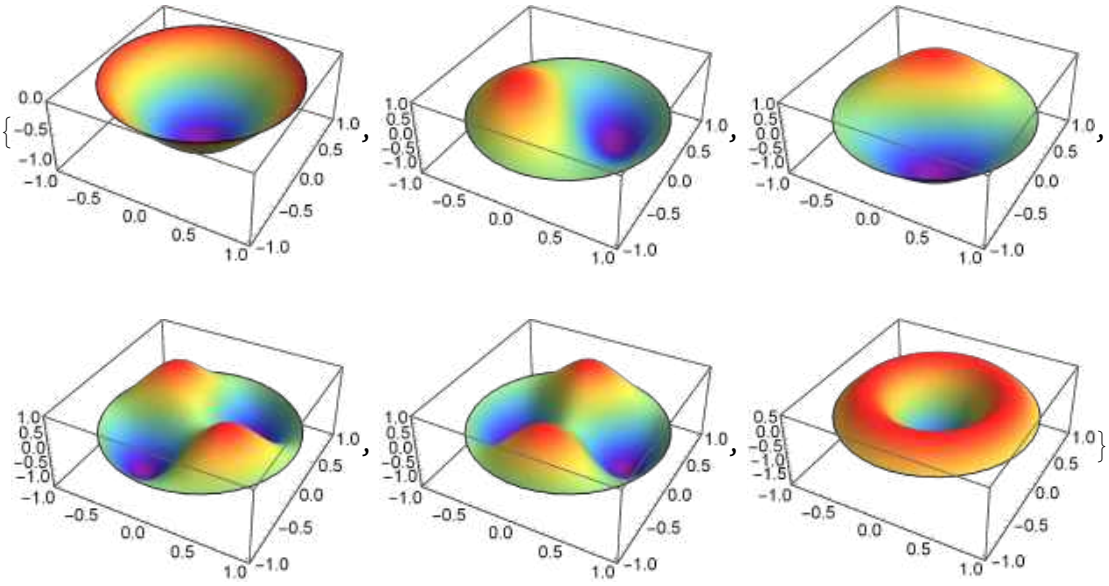
Out[•]=



In[•]:=
```
{vals, funs} = NDEigensystem[{-Laplacian[u[x, y], {x, y}],
    DirichletCondition[u[x, y] == 0, True]}, u[x, y], {x, y} ∈ Disk[], 6];
```

In[ ]:=
```
Table[Plot3D[i, {x, y} ∈ Disk[],
  ColorFunction → "Rainbow", Mesh → None, PlotPoints → 30], {i, funs}]
```

Out[ ]=

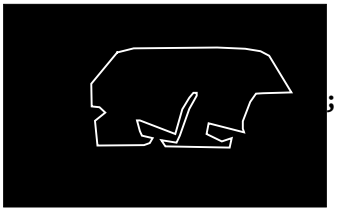


car acoustic distribution

In[ ]:=
```
img =
```

;

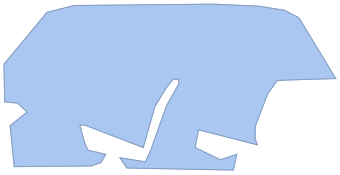
In[ ]:=
```
boundary =
```



;

In[ ]:=
```
region = BoundaryDiscretizeGraphics[boundary, ImageSize → Small]
```

Out[ ]=



In[ ]:=
```
{vals, funs} = NDEigensystem[{-Laplacian[u[x, y], {x, y}]}, u[x, y], {x, y} ∈ region, 6];
```

```
In[•]:= Show[img, ContourPlot[funs[[2]], {x, y} ∈ region, ColorFunction →
          Function[x, {Opacity[0.65], ColorData["Rainbow"][x]}]], ImageSize → Medium]
```

Out[•]=

other behaviours of NDSolve/ParametricNDSolve

using matrix/vector to index inside NDSolve

```
In[•]:= mat = {{1, 1}, {1, -1}};
```

```
In[•]:= sol = NDSolve[{x'[t] == mat.x[t], x[0] == {1, 2}}, x, {t, 0, 1}]
```

Out[•]= {{x → InterpolatingFunction[Domain: {{0., 1.}} Output dimensions: {2}]}}

```
In[•]:= x[0.5] /. sol
```

Out[•]= {{2.88875, 1.97846}}

putting constraints on the dependent var

```
In[•]:= NDSolve[{y'[t] == y[t], y[0] == 0.01}, y, {t, 0, 10}, DependentVariables → {y[t], 0, 1}]
```

... NDSolve: Dependent variable(s) y went out of the range specified by {y, 0, 1} at t = 4.605167779949835`.

Out[•]= {{y → InterpolatingFunction[Domain: {{0., 4.61}} Output: scalar]}}

```
In[•]:= NDSolve[{y'[t] == y[t], y[0] == 0.01}, y,
          {t, 0, 10}, DependentVariables → {y[t], 0, 1} :↦ Echo[t]]
```

```
» 4.60517
```

```
Out[•]= {{y → InterpolatingFunction[ ⊞  Domain: {{0., 10.}}
                                     Output: scalar ]}}
```

“WhenEvent” in ParametricNDSolve

```
(*trivial example to check its effect *)
```

```
In[•]:= sol = ParametricNDSolve[
          {x'[t] == Sin[t], x[0] == s, WhenEvent[x[t] == 1.1, x[t] → x[t] + 0.5]}, x, {t, 0, 1}, {s}]
```

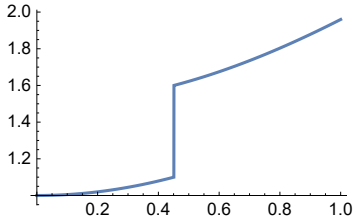
```
Out[•]= {x → ParametricFunction[ ⊞  Expression: x
                                 Parameters: {s} ]}
```

```
In[•]:= pfun = x[1] /. sol
```

```
Out[•]= InterpolatingFunction[ ⊞  Domain: {{0., 1.}}
                               Output: scalar ]
```

```
In[•]:= Plot[pfun[t], {t, 0, 1}, ImageSize → Small]
```

```
Out[•]=
```



“If” statement in NDSolve

In[390]:=
```
Manipulate[
 Module[{a = p[[1]], b = p[[-1]]},
  sol = NDSolve[{x'[t] == y[t], y'[t] == If[y[t] - 1 > 0, -1, 0.1 y[t] - x[t]],
     x[0] == a, y[0] == b}, {x, y}, {t, 0, 20}
   ];
  Row[{
    ParametricPlot[Evaluate[{x[t], y[t]} /. sol], {t, 0, 20}, ImageSize → Small],
    Plot[Evaluate[{x[t], y[t]} /. sol], {t, 0, 20}, ImageSize → Small]}
  ]
 ], {{p, {0., 0.53}}, Locator}, SaveDefinitions → True
]
```

Out[390]=