1 Eindimensionale Wärmeleitung

1.1 Die Wärmeleitungsgleichung

Set $T_0(\vec{x})$ die Temperaturverteilung in einem Körper M zur Zeit t_0. Sei M homogen, seine Wärmeleitfähigkeit κ also nicht ortsabhängig.

Durch Wärmeleitung wird sich diese
Temperaturverteilung $T(\vec{x}, t)$ mit der Zeit ändern. Dies beschreibt die Wärmeleitungsgleichung

$$\frac{\partial T(\vec{x}, t)}{\partial t} = \kappa \, \Delta T(\vec{x}, t)$$

mit dem Laplace-Operator Δ der Mannigfaltigkeit M.

Dies ist eine parabolische PDE und hat daher als typische Problemstellung das Anfangs-Randwertproblem: > Sei eine Anfangsverteilung $T_0(\vec{x})$ auf M gegeben. Sei eine Randwertverteilung $T_{\partial M}(\vec{x}, t)$ auf dem Rand ∂M gegeben. Unter gewissen Voraussetzungen an $M, T_0, T_{\partial M}$ hat die PDE eine eindeutige Lösung für alle Zeiten $t \geq t_0$.

1.2 Numerik in einer Dimension

Im eindimensionalen Fall (Wärmeleitung in einem Stab der Länge L) reduziert sich die Gleichung zu

$$\frac{\partial T(x, t)}{\partial t} = \kappa \, \frac{\partial^2 T(x, t)}{\partial x^2}$$

Die Anfangsbedingung $T(x, 0) = T_0(x)$ beschreibt die Temperaturverteilung längs des Stabes zu einer Zeit $t = 0$. Die Randwertbedingungen $T(0, t) = f_1(t)$ und $T(L, t) = f_2(t)$ beschreiben den zeitlichen Verlauf der von außen vorgegebenen Temperatur an den beiden Enden $x = 0$ und $x = L$.

Wir betrachten im Folgenden die zeitunabhängigen Randbedingungen $T(0, t) = T_1, \quad T(L, t) = T_2$.

1.3 Explizites Schema

Numerisch läßt sich dieses Problem durch Diskretisierung lösen. Wir wählen n gleichverteilte Stützstellen zwischen $x_0 = 0$ und $x_{n+1} = L$ auf dem Stab, der Gitterabstand ist $a = \frac{L}{n+1}$. Die Zeitentwicklung wird in Zeitschritten der Größe Δt berechnet. Die einfachste Diskretisierung der 2. Ableitung nach x ist

$$\frac{\partial^2}{\partial x^2} T(x)\bigg|_{x=x_i} \approx \frac{1}{a^2} \left(T_{i-1} - 2T_i + T_{i+1}\right)$$

Hier wurde die Abkürzung $T_i \coloneqq T(x_i)$ verwendet. Analog ist $T_{i,k} \coloneqq T(x_i, t_k)$.

Es gibt nun verschiedene Möglichkeiten, Diskretisierungen der Zeit- und Ortsableitungen zu kombinieren. Der einfachste Fall ist das sogenannte explizite Schema:

$$\frac{T_{i,k+1} - T_{i,k}}{\Delta t} = \kappa \frac{1}{a^2} \left(T_{i-1,k} - 2T_{i,k} + T_{i+1,k}\right)$$

Dieses Gleichungssystem läßt sich sofort nach den $T_{i,k+1}$ auflösen. Mit $z = \frac{\kappa \Delta t}{a^2}$ erhält man

$$T_{i,k+1} = z(T_{i-1,k} - 2T_{i,k} + T_{i+1,k}) + T_{i,k}$$

Mit Hilfe der Matrix

$$= \begin{pmatrix} -2 & 1 & 0 & ... & 0 \\ 1 & -2 & 1 & ... & 0 \\ 0 & 1 & -2 & ... & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & ... & 1 & -2 & 1 \\ 0 & ... & 0 & 1 & -2 \end{pmatrix}$$

(dem *diskreten Laplaceoperator in einer Dimension*) kann das in Vektorschreibweise auch als

$$\vec{T}_{k+1} = (\mathbf{1} + z\,)\vec{T}_k$$

geschrieben werden, wobei $\mathbf{1}$ die $n \times n$ Einheitsmatrix und $\vec{T}_k$ der Vektor der längs des Stabes gemessenen Temperaturen zur Zeit t
ist. Damit ist die Berechnung der Temperaturverteilung $\vec{T}_k = \{T(i,k), i = 1 \ldots n\}$ auf dem Stab zu einem Zeitpunkt k aus der Verteilung zur Zeit 0 auf eine einfache Reihe von k Matrixmultiplikationen zurückgeführt.

Leider hat dieser einfache Algorithmus, das "explizite Schema", gravierende Nachteile. Die Diskretisierungsfehler sind relativ groß (von der Ordnung Δt) und der Algorithmus ist für $z > \frac{1}{2}$ instabil, d.h. die berechnete Zeitentwicklung entfernt sich immer mehr von der tatsächlichen Lösung.

1.4 Crank-Nicholson Schema

Im Gegensatz dazu ist das Crank-Nicholson Schema stabil und seine Diskretisierungsfehler sind nur von der Ordnung $(\Delta t)^2$. Hierbei verwendet man auf der rechten Seite der diskreten Wärmeleitungsgleichung den Mittelwert der räumlichen Ableitungen zur Zeit k und zur Zeit $k+1$:

$$\frac{T_{i,k+1} - T_{i,k}}{\Delta t} = \frac{\kappa}{2}\left[\frac{T_{i-1,k+1} - 2T_{i,k+1} + T_{i+1,k+1}}{a^2} + \frac{T_{i-1,k} - 2T_{i,k} + T_{i+1,k}}{a^2}\right]$$

In Vektorschreibweise führt das auf die Gleichung

$$\left(\mathbf{1} - \frac{z}{2}\,\right)\vec{T}_{k+1} = \left(\mathbf{1} + \frac{z}{2}\,\right)\vec{T}_k$$

$(\mathbf{1} - \frac{z}{2}\,)$ ist, wie selbst, eine Tridiagonal-Matrix, so daß die Lösung dieses linearen Gleichungssystems numerisch nicht sehr aufwendig ist.

1.4.1 Randbedingungen

- Implementierung von Randbedingungen in numerischen Solvern kann nichttrivial sein.
- Beschränkung auf zeitunabhängige Randwerte $T(0,t) = T_1, \quad T(L,t) = T_2$ erlaubt Trick:
  - Diskreter Laplacian Δ wird so definiert, dass er für die Randpunke verschwindet.
  - Dadurch bleiben die Randwerte automatisch zeitlich konstant.

$$= \begin{pmatrix} 0 & 0 & 0 & \ldots & 0 & 0 \\ 1 & -2 & 1 & \ldots & 0 & 0 \\ 0 & 1 & -2 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \ldots & 0 & 1 & -2 & 1 \\ 0 & \ldots & 0 & 0 & 0 & 0 \end{pmatrix}$$

[1]:
```
using LinearAlgebra
```

[2]:
```
"""
1-dim discrete Laplacian for n points, vanishing at first and last point
in order to preserve time independet Dirichlet boundary conditions
"""
function Δ(n)
    dl = [ fill(1., n-2); [0] ]         # lower subdiag
    d  = [ [0]; fill(-2., n-2); [0] ]   # diag
    du = [ [0]; fill(1., n-2) ]         # upper subdiag
    Δ  = Tridiagonal(dl, d, du)
    return Δ
end
```

[2]:
```
Δ
```

[3]:
```
Δ(7)
```

[3]:
```
7×7 Tridiagonal{Float64, Vector{Float64}}:
 0.0   0.0    ⋅     ⋅     ⋅     ⋅    ⋅
 1.0  -2.0   1.0    ⋅     ⋅     ⋅    ⋅
  ⋅    1.0  -2.0   1.0    ⋅     ⋅    ⋅
  ⋅     ⋅    1.0  -2.0   1.0    ⋅    ⋅
  ⋅     ⋅     ⋅    1.0  -2.0   1.0   ⋅
  ⋅     ⋅     ⋅     ⋅    1.0  -2.0  1.0
  ⋅     ⋅     ⋅     ⋅     ⋅    0.0  0.0
```

[4]:
```
"""
Explizites Schema zur 1D Wärmeleitungsgleichung. Ein Startvektor der Anfangsverteilung wird
nsteps Schritte in der Zeit entwickelt, die Randwerte werden festgehalten.
Zurückgegeben wird die Matrix T[i,k]  mit Ortsindex i und Zeitindex k.
"""
function explicit_steps(startvec, z, nsteps)
    n, = size(startvec)
    M = I + z .* Δ(n)
    res = copy(startvec)

    for i = 1:nsteps
        next = M * res[:, end]
        res  = hcat(res, next)         # anfuegen der neuberechneten Spalte
    end
    return res
end
```

[4]:
```
explicit_steps
```

[5]:
```
# einfacher Test
explicit_steps([.3,.3,.7,.7,.3,.3], .4, 8)
```

[5]:
```
6×9 Matrix{Float64}:
 0.3  0.3   0.3    0.3     0.3      0.3       0.3       0.3       0.3
 0.3  0.46  0.428  0.4088  0.39216  0.37808   0.36615   0.356044  0.347481
 0.7  0.54  0.508  0.476   0.44912  0.426336  0.407034  0.39068   0.376826
 0.7  0.54  0.508  0.476   0.44912  0.426336  0.407034  0.39068   0.376826
 0.3  0.46  0.428  0.4088  0.39216  0.37808   0.36615   0.356044  0.347481
 0.3  0.3   0.3    0.3     0.3      0.3       0.3       0.3       0.3
```

1.5 Visualisierung

- `PlotlyJS` ermöglicht interaktive Visualisierungen (Drehen, Zoom, …) im Browser durch den Einsatz von Javascript.
- Mit dem folgenden 'Hack' überlebt die Interaktivität die Konvertierung vom Jupyter-Notebook nach HTML.
- Leider funktioniert die Konvertierung Notebook -> PDF nicht. Darum machen wir zusätzlich einen PNG-snapshot der Grafiken.

[6]:
```
using PlotlyJS
```

```
WebIO._IJuliaInit()
```

[7]:
```
## hack fuer HTML-Export, Quelle:
## https://discourse.julialang.org/t/
  ↪pluto-plots-with-plotlybase-plotlyjs-with-working-static-html-export/61730


using HypertextLiteral

function Base.show(io::IO, mimetype::MIME"text/html", p::PlotlyBase.Plot)
    show(io,mimetype,@htl("""
<script src='https://cdn.plot.ly/plotly-latest.min.js'></script>
```

```
    <div style="height: auto">
        <script id=plotly-show>
            const PLOT = this ?? document.createElement("div");
            (this == null ? Plotly.newPlot : Plotly.react)(PLOT,$(HypertextLiteral.
 ↪JavaScript(json(p))));
            return PLOT
        </script>
    </div>
"""))
end
```

(1) eine lineare Temperaturverteilung: die beiden Stabenden werden auf 0 bzw 100 Grad gehalten; 200 Stützstellen

[8]:
```
T0 = LinRange(0,100,200)
```

[8]:
```
200-element LinRange{Float64, Int64}:
 0.0,0.502513,1.00503,1.50754,2.01005,…,97.9899,98.4925,98.995,99.4975,100.0
```

[9]:
```
H = explicit_steps(T0, .3, 300)
```

[9]:
```
200×301 Matrix{Float64}:
   0.0         0.0         0.0       …    0.0         0.0         0.0
   0.502513    0.502513    0.502513       0.502513    0.502513    0.502513
   1.00503     1.00503     1.00503        1.00503     1.00503     1.00503
   1.50754     1.50754     1.50754        1.50754     1.50754     1.50754
   2.01005     2.01005     2.01005        2.01005     2.01005     2.01005
   2.51256     2.51256     2.51256   …    2.51256     2.51256     2.51256
   3.01508     3.01508     3.01508        3.01508     3.01508     3.01508
   3.51759     3.51759     3.51759        3.51759     3.51759     3.51759
   4.0201      4.0201      4.0201         4.0201      4.0201      4.0201
   4.52261     4.52261     4.52261        4.52261     4.52261     4.52261
   5.02513     5.02513     5.02513   …    5.02513     5.02513     5.02513
   5.52764     5.52764     5.52764        5.52764     5.52764     5.52764
   6.03015     6.03015     6.03015        6.03015     6.03015     6.03015
   ⋮                                  ⋱                          ⋮
  94.4724     94.4724     94.4724        94.4724     94.4724     94.4724
  94.9749     94.9749     94.9749        94.9749     94.9749     94.9749
  95.4774     95.4774     95.4774   …    95.4774     95.4774     95.4774
  95.9799     95.9799     95.9799        95.9799     95.9799     95.9799
  96.4824     96.4824     96.4824        96.4824     96.4824     96.4824
  96.9849     96.9849     96.9849        96.9849     96.9849     96.9849
  97.4874     97.4874     97.4874        97.4874     97.4874     97.4874
  97.9899     97.9899     97.9899   …    97.9899     97.9899     97.9899
  98.4925     98.4925     98.4925        98.4925     98.4925     98.4925
  98.995      98.995      98.995         98.995      98.995      98.995
  99.4975     99.4975     99.4975        99.4975     99.4975     99.4975
 100.0       100.0       100.0          100.0       100.0       100.0
```

[10]:
```
p1 = Plot(surface(z=H))
```

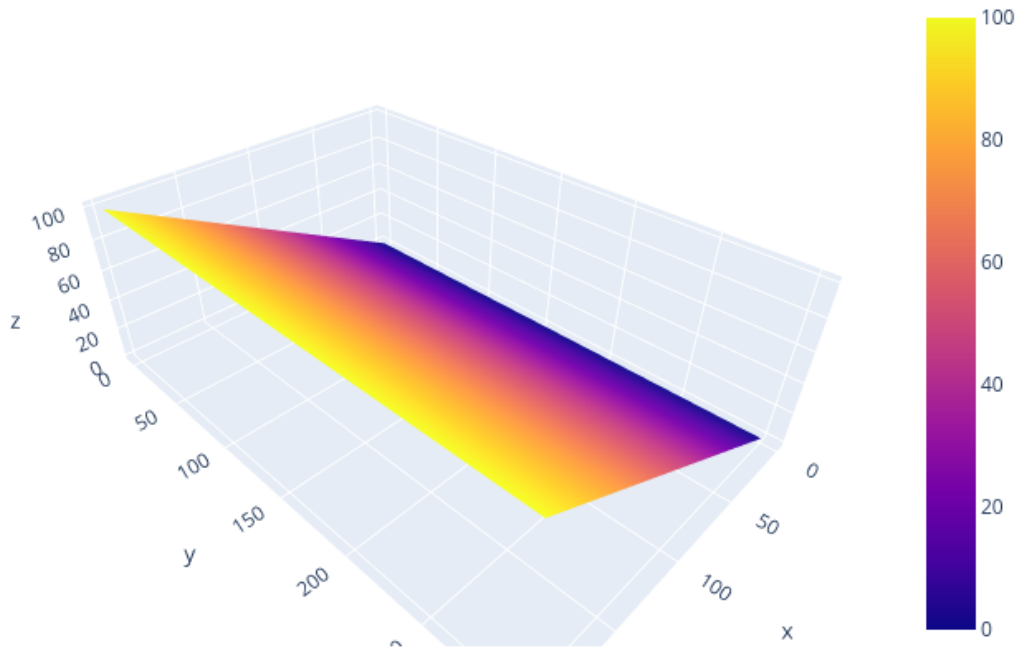
[10]:
```
data: [
  "surface with fields type and z"
]

layout: "layout with fields margin and template"
```

[10]:
```
savefig(p1, "p1.png")
```

[10]:
```
"p1.png"
```

hier das PNG für die PDF-Konvertierung:



- Math: triviale Lösung, $\Delta T = \frac{\partial^2 T}{\partial x^2} = 0 \quad \Rightarrow \frac{\partial T}{\partial t} = 0$; Lösung ist stationär (zeitunabhängig).
- Physik:
  - ständiger Wärmetransport vom heißen Ende des Stabs zum kalten Ende.
  - Die Randbed. "$T_1 = 0$ für alle Zeiten" und "$T_2 = 100$ für alle Zeiten" ist eine physikalische Idealisierung:
  - Das jeweilige Wärmebad ist unendlich groß/hat unendliche Wärmekapazität, d.h., beliebiger Zu- oder Abfluß von Wärme führt zu keiner Temperaturänderung.

[11]:
```
T0 = ((x-> x^4) ∘ sin).(0:.04:2π)
```

[11]:
```
158-element Vector{Float64}:
 0.0
 2.5572706436761415e-6
 4.078557249163394e-5
 0.00020537792139572187
 0.0006442606943161432
 0.001557841666953136
 0.003192536868694855
 0.005832750277880962
 0.00979152226184813
 0.015400099021820692
 0.022996705038756202
 0.032914823039099886
 0.04547130056990937
 ⋮
 0.0338140301063203
 0.023697122823874928
 0.015927725217493763
 0.010173243361530223
 0.00609539655087857
 0.0033619629732698983
 0.0016580578800919176
```

```
0.0006966382944063128
0.00022795016786367837
4.766312908956853e-5
3.4737939504564656e-6
1.0294430936302562e-10
```

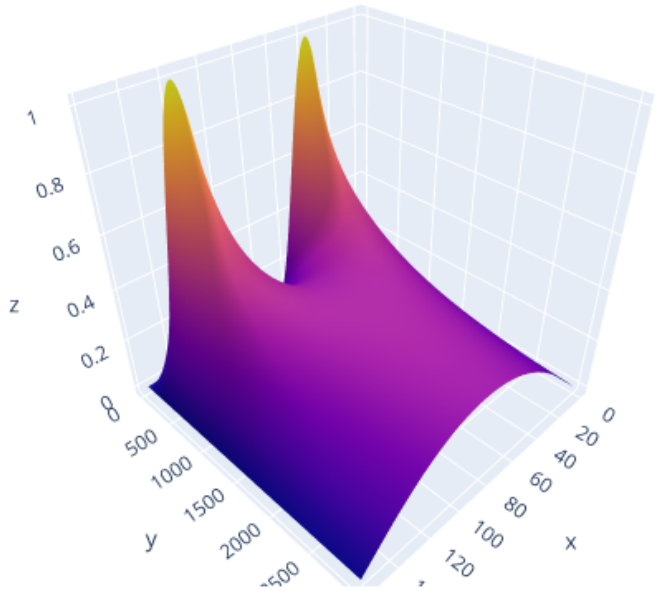
[12]:
```
H = explicit_steps(T0, .4, 3000)
p2 = Plot(surface(z=H))
```

[12]:
```
data: [
  "surface with fields type and z"
]

layout: "layout with fields margin and template"
```

[23]:
```
savefig(p2,"p2.png")
```

[23]:
```
"p2.png"
```



Hier das PNG für die PDF-Konvertierung:

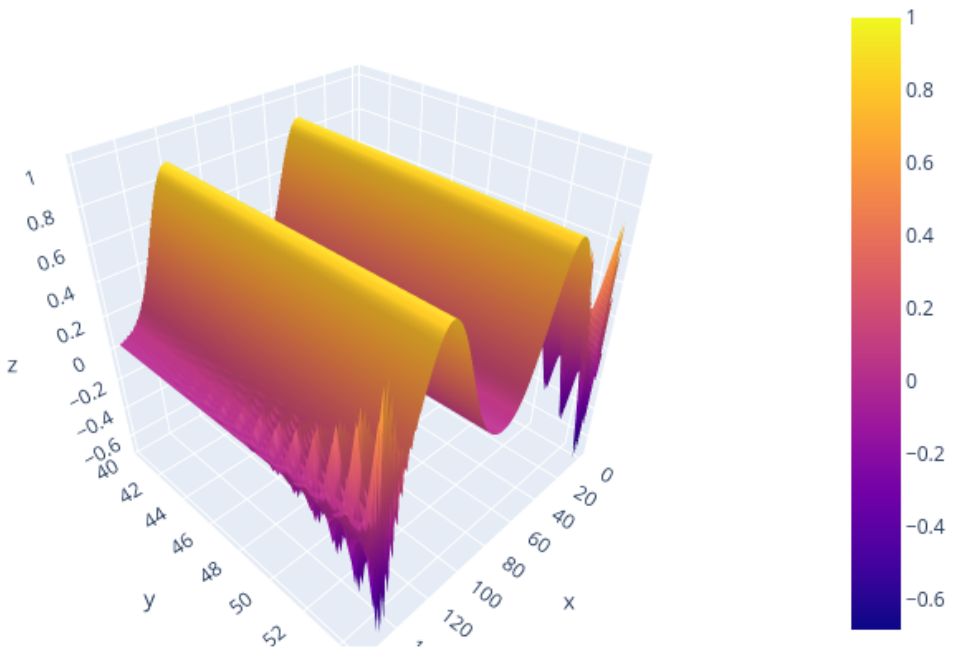
[13]:
```
H = explicit_steps(T0, .6, 55)
p3 = Plot(surface(z=H), Layout(scene_yaxis_range=[40, 55]))
```

[13]:
```
data: [
  "surface with fields type and z"
]

layout: "layout with fields margin, scene, and template"
```

[41]:
```
savefig(p3,"p3.png")
```

[41]:
```
"p3.png"
```



[14]:
```
"""
Crank-Nicolson Schema zur 1D Wärmeleitungsgleichung. Ein Startvektor der Anfangsverteilung wird
nsteps Schritte in der Zeit entwickelt, die Randwerte werden festgehalten.
Zurückgegeben wird die Matrix T[i,k]  mit Ortsindex i und Zeitindex k.
"""
function cn_steps(startvec, z, nsteps)
    n, = size(startvec)
    M1 = I + 0.5 * z .* Δ(n)
    M2 = I - 0.5 * z .* Δ(n)
    M2f = factorize(M2)
    res = copy(startvec)

    for i = 1:nsteps
        rhs = M1 * res[:,end]
        next = M2f\rhs
        res  = hcat(res, next)         # anfuegen der neuberechneten Spalte
    end
    return res
end
```

[14]:
```
cn_steps
```

[15]:
```
function heatflow(startvec, z, nsteps)
    H = cn_steps(startvec, z, nsteps)
    p = Plot(surface(z=H))
    return p
end
```

[15]:
```
heatflow (generic function with 1 method)
```

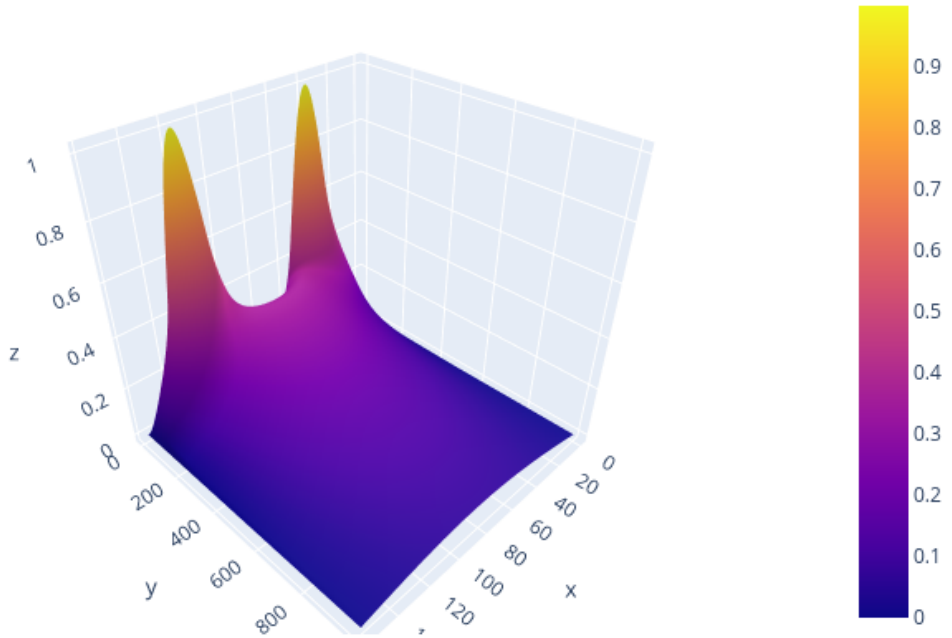
[16]:
```
p4 = heatflow(T0, 5, 1000)
```

[16]:
```
data: [
  "surface with fields type and z"
]
```

```
    layout: "layout with fields margin and template"
```

[50]:
```
savefig(p4, "p4.png")
```

[50]:
```
"p4.png"
```



[17]:
```
T0 = append!(fill(0., 50), fill(1.,3), fill(0.,50), collect(0:.02:.3))
```

[17]:
```
119-element Vector{Float64}:
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 0.0
 ⋮
 0.08
 0.1
 0.12
 0.14
 0.16
 0.18
 0.2
 0.22
```

```
0.24
0.26
0.28
0.3
```

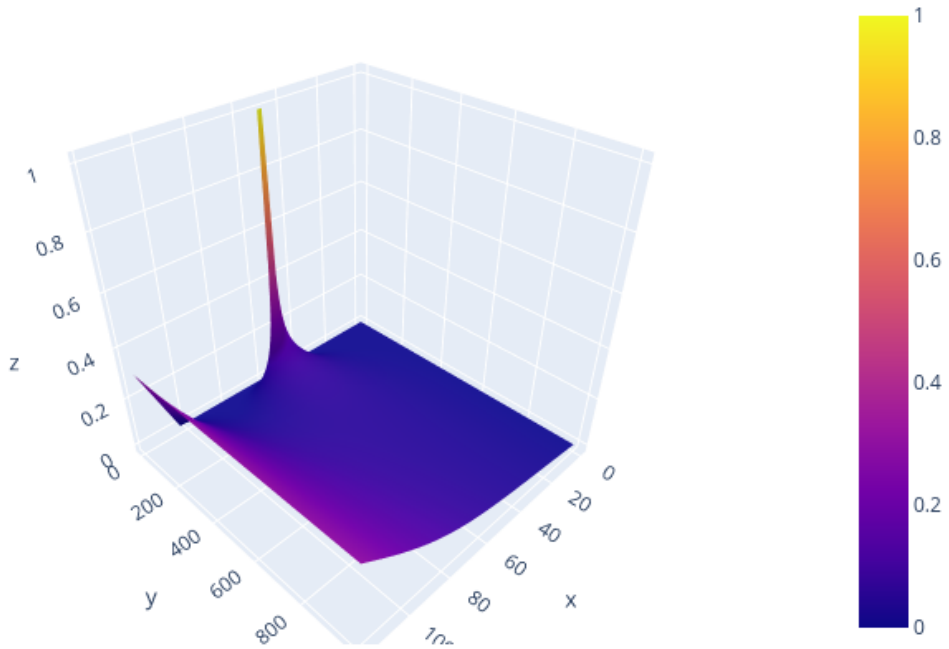
[18]:
```
p5 = heatflow(T0, .7, 1000)
```

[18]:
```
data: [
  "surface with fields type and z"
]

layout: "layout with fields margin and template"
```

[92]:
```
savefig(p5, "p5.png")
```

[92]:
```
"p5.png"
```



[ ]:
```

```