```
begin
    using Pkg
    Pkg.activate(mktempdir())
    Pkg.add("PyPlot")
    Pkg.add("PlutoUI")
    Pkg.add("DataFrames")
    using PlutoUI
    using PyPlot
    using DataFrames
end
```

Matrices from partial differential equations

As we focus in this course on partial differential equations, we need discuss matrices which evolve from the discretization of PDEs.

- Are there any structural or numerical patterns in these matrices we can take advantage of with regard to memory and time complexity when solving linear systems ?

In this lecture we introduce a relatively simple "drosophila" problem which we will use do discuss these issues.

For the start we use simple structured disceretization grids and a finite difference approach to the discretization. Later, this will be generalized to more general grids and to finite element and finite volume discretization methods.

Heat conduction in a one-dimensional rod

- Heat source $f(x)$
- v_L, v_R: ambient temperatures
- α: boundary heat transfer coefficient
- Second order boundary value problem in $\Omega = [0, 1]$:

$$\begin{cases} -u''(x) & = f(x) \text{ in } \Omega \\ -u'(0) + \alpha(u(0) - v_L) & = 0 \\ u'(1) + \alpha(u(1) - v_R) & = 0 \end{cases}$$

- The solution u describes the equilibrium temperature distribution. Behind the second derivative is Fouriers law and the continuity equation
- In math, the boundary conditions are called "Robin" or "third kind". They describe a heat in/outflux proportional to the difference between rod end temperature and ambient temperature
- Fix a number of discretization points N
- Let $h = \frac{1}{N-1}$
- Let $x_i = (i-1)h \;\; i = 1 \ldots N$ be discretization points

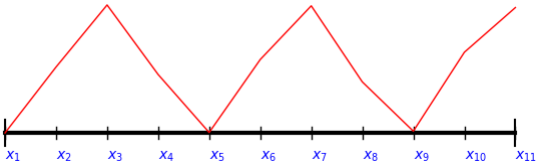
```
N = 11
```
```
N=11
```

```
plotgrid (generic function with 1 method)
```


```
plotgrid(N)
```

Finite difference approximation

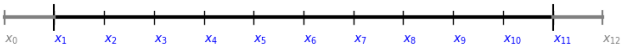
We can approximate continuous functions f by piecewise linear functions defined by the values $f_i = f(x_i)$. Using more points yields a better approximation:




```
plotgrid(N,func=x->0.5*sin(8*x)^2)
```

- Let u_i approximations for $u(x_i)$ and $f_i = f(x_i)$
- We can use a finite difference approximation to approximate $u'(x_{i+\frac{1}{2}}) \approx \frac{u_{i+1} - u_i}{h}$
- Same approach for second derivative: $u''(x_i) = \frac{u'(x_{i+\frac{1}{2}}) - u'(x_{i-\frac{1}{2}})}{h}$
- Finite difference approximation of the PDE:

$$\begin{aligned} -u'(0) + \alpha(u(0) - v_L) &\approx \frac{1}{2h}(u_0 - u_2) + \alpha(u_1 - v_L) = 0 \\ -u''(x_i) - f(x_i) &\approx \frac{-u_{i+1} + 2u_i - u_{i-1}}{h^2} - f_i = 0 \qquad (i = 1 \dots N)\ (*) \\ u'(1) + \alpha(u(1) - v_R) &\approx \frac{1}{2h}(u_{N+1} - u_{N-1}) + \alpha(u_N - v_R) = 0 \end{aligned}$$




```
plotgrid(N,mirror=true)
```

- Here, we introduced "mirror values" u_0 and u_{N+1} in order to approximate the boundary conditions accurately, such that the finite difference formulas used to approximate $u'(0)$ or $u'(1)$ are centered around these values.
- After rearranging, these values can be expressed via the boundary conditions:

$$\begin{aligned} u_0 &= u_2 + 2h\alpha(u_1 - v_L) \\ u_{N+1} &= u_{N-1} + 2h\alpha(u_N - v_L) \end{aligned}$$

- Finally, they can be replaced in $(*)$

Then, the system after multiplying by h is reduced to:

$$\begin{aligned} \frac{1}{h}((1 + h\alpha)u_1 - u_2) &= \frac{h}{2}f_1 + \alpha v_L \\ \frac{1}{h}(-u_{i+1} + 2u_i - u_{i-1}) &= hf_i \qquad (i = 2 \dots N-1) \\ \frac{1}{h}((1 + h\alpha)u_N - u_{N-1}) &= \frac{h}{2}f_N + \alpha v_R \end{aligned}$$

The resulting discretization matrix is

$$A = \begin{pmatrix} \alpha+\frac{1}{h} & -\frac{1}{h} & & & & & \\ -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & & \\ & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & & \\ & \ddots & \ddots & \ddots & \ddots & & \\ & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & & \\ & & & -\frac{1}{h} & \frac{2}{h} & -\frac{1}{h} & \\ & & & & -\frac{1}{h} & \frac{1}{h}+\alpha \end{pmatrix}$$

Outside of the three diagonals, the entries are zero.

The right hand side is:

$$\begin{pmatrix} \frac{h}{2}f_1 + \alpha v_L \\ hf_2 \\ hf_3 \\ \vdots \\ hf_{N-2} \\ hf_{N-1} \\ \frac{h}{2}f_N + \alpha v_R \end{pmatrix}$$

Let us define functions assembling these:

heatmatrix1d (generic function with 1 method)

```
function heatmatrix1d(N;α=1)
    A=zeros(N,N)
    h=1/(N-1)
    A[1,1]=1/h+α
    for i=2:N-1
        A[i,i]=2/h
    end
    for i=1:N-1
        A[i,i+1]=-1/h
    end
    for i=2:N
        A[i,i-1]=-1/h
    end
    A[N,N]=1/h+α
    A
end
```

heatrhs1d (generic function with 1 method)

```
function heatrhs1d(N;vl=0,vr=0,func=x->0,α=1)
    h=1/(N-1)
    F=zeros(N)
    F[1]=h/2*func(0)+α*vl
    for i=2:N-1
        F[i]=h*func((i-1)*h)
    end
    F[N]=h/2*func(1)+α*vr
    F
end
```

α = 100

```
α=100
```

N1 = 10000

```
N1=10000
```

A = 10000×10000 Array{Float64,2}:

```
 10099.0  -9999.0      0.0      0.0      0.0  …      0.0      0.0      0.0      0.0
 -9999.0  19998.0  -9999.0      0.0      0.0         0.0      0.0      0.0      0.0
     0.0  -9999.0  19998.0  -9999.0      0.0         0.0      0.0      0.0      0.0
     0.0      0.0  -9999.0  19998.0  -9999.0         0.0      0.0      0.0      0.0
     0.0      0.0      0.0  -9999.0  19998.0         0.0      0.0      0.0      0.0
     0.0      0.0      0.0      0.0  -9999.0  …      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
     ⋮                                    ⋱
     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      0.0  …  -9999.0      0.0      0.0      0.0
     0.0      0.0      0.0      0.0      0.0      19998.0  -9999.0      0.0      0.0
     0.0      0.0      0.0      0.0      0.0      -9999.0  19998.0  -9999.0      0.0
```

```
 0.0   0.0   0.0   0.0   0.0     0.0  -9999.0  19998.0  -9999.0
 0.0   0.0   0.0   0.0   0.0     0.0      0.0  -9999.0  10099.0
```

```
A=heatmatrix1d(N1,α=α)
```

b =

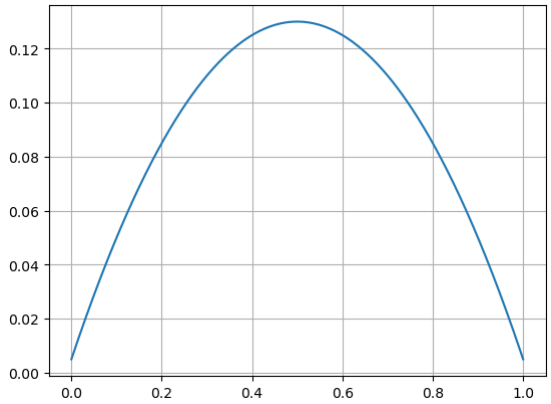
Float64[5.0005e-5, 0.00010001, 0.00010001, 0.00010001, 0.00010001, 0.00010001, 0.0001

```
b=heatrhs1d(N1,func=x->1,α=α)
```

u =

Float64[0.005, 0.00505, 0.00509999, 0.00514997, 0.00519994, 0.0052499, 0.00529985, 0

```
u=A\b
```




```
begin
    clf()
    plot(collect(0:1/(N1-1):1),u)
    grid()
    gcf()
end
```

For this example, we created an $N \times N$ matrix where all entries outside of the main diagonal and the two adjacent ones are zero:

- Fraction of nonzero entries: 0.00029998
- Ratio of nonzero entries to number of unknowns: 2.9998
- In fact, this matrix has $O(N)$ nonzero entries.

2D heat conduction

Just pose the heat problem in a 2D domain $\Omega = (0,1) \times (0,1)$:

$$\begin{cases} -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} & = f(x,y) \text{ in } \Omega \\ \frac{\partial u}{\partial n} + \alpha(u - v) & = 0 \text{ on } \partial\Omega \end{cases}$$

We use 2D regular discretization $n \times n$ grid with grid points $x_{ij} = ((i-1)h, (j-1)h)$. The finite difference approximation yields:

$$\frac{-u_{i-1,j} - u_{i,j-1} + 4u_{ij} - u_{i+1,j} - u_{i,j+1}}{h^2} = f_{ij}$$

This just comes from summing up the 1D finite difference formula for the x and y directions.

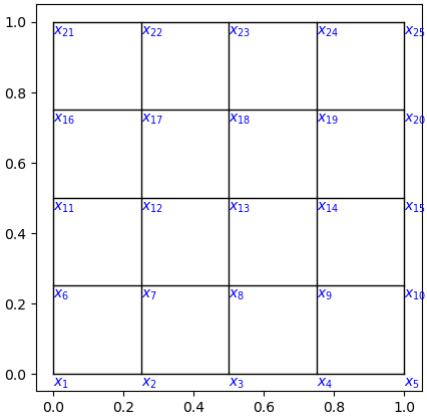
We do not discuss the boundary conditions here.

The $n \times n$ grid leads to an $n^2 \times n^2$ matrix!

```
plotgrid2d (generic function with 1 method)
```

```
function plotgrid2d(N;text=true, func=nothing)
    clf()
    ax=PyPlot.axes(aspect=1)
    x=[(i-1)/(N-1) for i=1:N]
    y=[(i-1)/(N-1) for i=1:N]

    for i=1:N
        plot([x[i],x[i]],[0,1],linewidth=1,color="k")
        plot([0,1],[y[i],y[i]],linewidth=1,color="k")
    end
    if func!=nothing
        f=[func(x[i],y[j]) for i=1:N, j=1:N]
        contourf(x,y,f,cmap="hot")
    end
    if text
      ij=1
      for j=1:N
        for i=1:N
          ax.text(x[i],y[j]-0.035,"\$x_{$(ij)}\$",fontsize=10,color=:blue)
            ij=ij+1
        end
      end
    end
    fig=PyPlot.gcf()
    fig.set_size_inches(5,5)
    fig
end
```




```
plotgrid2d(5)
```

Matrix and right hand side assembly inspired by the finite volume method which will be covered later in the course. The result is the same as for the finite difference method with the mirror trick for the boundary condition.

```
heatmatrix2d (generic function with 1 method)
```

```
function heatmatrix2d(n;α=1)
    function update_pair(A,v,i,j)
        A[i,j]+=-v
        A[j,i]+=-v
        A[i,i]+=v
        A[j,j]+=v
    end
    N=n^2
    h=1.0/(n-1)
    A=zeros(N,N)
    l=1
        for j=1:n
            for i=1:n
                if i<n
                    update_pair(A,1.0,l,l+1)
                end
                if i==1|| i==n
                    A[l,l]+=α
```

```
                end
                if j<n
                    update_pair(A,1,l,l+n)
                end
                if j==1|| j==n
                    A[l,l]+=α
                end
                l=l+1
            end
        end
    A
end
```

heatrhs2d (generic function with 1 method)

```
function heatrhs2d(n; rhs=(x,y)->0,bc=(x,y)->0,α=1.0)
    h=1.0/(n-1)
    x=collect(0:h:1)
    y=collect(0:h:1)
    N=n^2
    f=zeros(N)
    for i=1:n-1
        for j=1:n-1
            ij=(j-1)*n+i
            f[ij]+=h^2/4*rhs(x[i],y[j])
            f[ij+1]+=h^2/4*rhs(x[i+1],y[j])
            f[ij+n]+=h^2/4*rhs(x[i],y[j+1])
            f[ij+n+1]+=h^2/4*rhs(x[i+1],y[j+1])
        end
    end

    for i=1:n
        ij=i
        fac=h
        if i==1 || i==n
            fac=h/2
        end
        f[ij]+=fac*α*bc(x[i],0)
        ij=i+(n-1)*n
        f[ij]+=fac*α*bc(x[i],1)
    end
    for j=1:n
        fac=h
        if j==1 || j==n
            fac=h/2
        end
        ij=1+(j-1)*n
        f[ij]+=fac*α*bc(0,y[j])
        ij=n+(j-1)*n
        f[ij]+=fac*α*bc(1,y[j])
    end
    f
end
```

n = 5

```
n=5
```

b2 =

```
Float64[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.03125, -0.0441942, 0.03125, 8.11834e-18, 0
```

```
b2=heatrhs2d(n,rhs=(x,y)->sin(3*π*x)*sin(3*π*y),α=α)
```

A2 =

```
25×25 Array{Float64,2}:
 202.0   -1.0    0.0    0.0    0.0   -1.0   0.0  …    0.0    0.0    0.0    0.0    0.0
  -1.0  103.0   -1.0    0.0    0.0    0.0  -1.0       0.0    0.0    0.0    0.0    0.0
   0.0   -1.0  103.0   -1.0    0.0    0.0   0.0       0.0    0.0    0.0    0.0    0.0
   0.0    0.0   -1.0  103.0   -1.0    0.0   0.0       0.0    0.0    0.0    0.0    0.0
   0.0    0.0    0.0   -1.0  202.0    0.0   0.0       0.0    0.0    0.0    0.0    0.0
  -1.0    0.0    0.0    0.0    0.0  103.0  -1.0  …    0.0    0.0    0.0    0.0    0.0
   0.0   -1.0    0.0    0.0    0.0   -1.0   4.0       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    0.0    0.0    0.0   -1.0
   0.0    0.0    0.0    0.0    0.0    0.0   0.0  …  202.0   -1.0    0.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0   0.0      -1.0  103.0   -1.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0   0.0       0.0   -1.0  103.0   -1.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0   0.0       0.0    0.0   -1.0  103.0   -1.0
   0.0    0.0    0.0    0.0    0.0    0.0   0.0       0.0    0.0    0.0   -1.0  202.0
```

```
A2=heatmatrix2d(n,α=α)
```

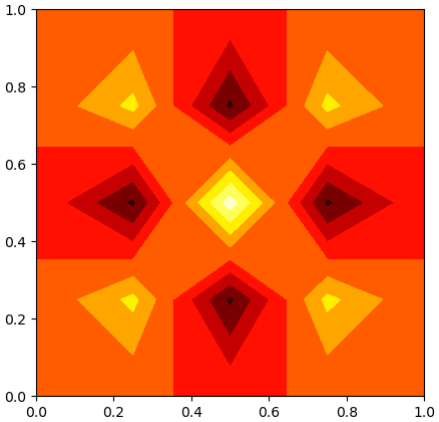
In order to inspect the matrix, we can turn it into a DataFrame, which can be browsed.

	x1	x2	x3	x4	x5	x6	x7	x8	mor
1	202.0	-1.0	0.0	0.0	0.0	-1.0	0.0	0.0	
2	-1.0	103.0	-1.0	0.0	0.0	0.0	-1.0	0.0	
3	0.0	-1.0	103.0	-1.0	0.0	0.0	0.0	-1.0	
4	0.0	0.0	-1.0	103.0	-1.0	0.0	0.0	0.0	
5	0.0	0.0	0.0	-1.0	202.0	0.0	0.0	0.0	
6	-1.0	0.0	0.0	0.0	0.0	103.0	-1.0	0.0	
7	0.0	-1.0	0.0	0.0	0.0	-1.0	4.0	-1.0	
8	0.0	0.0	-1.0	0.0	0.0	0.0	-1.0	4.0	
9	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	-1.0	
10	0.0	0.0	0.0	0.0	-1.0	0.0	0.0	0.0	

```
DataFrame(A2)
```

```
u2 =
  Float64[4.35673e-7,  4.4003e-5,  -6.20691e-5,  4.4003e-5,  4.35673e-7,  4.4003e-5,  0.00459
```

```
u2=A2\b2
```




```
begin
    clf()
    h=1.0/(n-1)
    x=collect(0:h:1)
    y=collect(0:h:1)

    contourf(x,y,reshape(u2,n,n),cmap="hot")
    fig=gcf()
    fig.set_size_inches(5,5)
    fig
end
```

In order to achieve this, we stored a matrix which has only five nonzero diagonals as a full $N \times N$ matrix, where $N = n^2$:

- Fraction of nonzero entries: 0.168
- Ratio of nonzero entries to number of unknowns: 4.2
- In fact, this matrix has $O(N)$ nonzero entries.

... there must be a better way!