3.1 Parabolic model problem

We are solving the unsteady heat equation

$$\text{find } u : [0,T] \to H^1_{0,D} \quad \int_\Omega \partial_t uv + \int_\Omega \nabla u \nabla v + b \cdot \nabla uv = \int fv \quad \forall v \in H^1_{0,D}, \quad u(t = ($$

with a suitable advective field b (the wind).

In [ ]:
```
from ngsolve import *
from math import pi
from netgen.geom2d import SplineGeometry
from ngsolve.webgui import Draw
```

- Geometry: $(-1, 1)^2$
- Dirichlet boundaries everywhere
- Mesh

In [ ]:
```
geo = SplineGeometry()
geo.AddRectangle( (-1, -1), (1, 1),
                  bcs = ("bottom", "right", "top", "left"))
mesh = Mesh( geo.GenerateMesh(maxh=0.25))
Draw(mesh)
fes = H1(mesh, order=3, dirichlet="bottom|right|left|top")

u,v = fes.TnT()

time = 0.0
dt = 0.001
```

We define the field b (the wind) as

$$b(x, y) := (2y(1 - x^2), -2x(1 - y^2)).$$

In [ ]:
```
b = CoefficientFunction((2*y*(1-x*x),-2*x*(1-y*y)))
Draw(b,mesh,"wind")
from ngsolve.internal import visoptions
visoptions.scalfunction = "wind:0"
```

- bilinear forms for
  - convection-diffusion stiffness and
  - mass matrix seperately.
- non-symmetric memory layout for the mass matrix so that a and m have the same sparsity pattern.

In [ ]:
```
a = BilinearForm(fes, symmetric=False)
a += 0.01*grad(u)*grad(v)*dx + b*grad(u)*v*dx
a.Assemble()
```

```
m = BilinearForm(fes, symmetric=False)
m += u*v*dx
m.Assemble()
```

Implicit Euler method

$$\underbrace{(M + \Delta t A)}_{M^*} u^{n+1} = M u^n + \Delta t f^{n+1}$$

or in an incremental form:

$$M^*(u^{n+1} - u^n) = -\Delta t A u^n + \Delta t f^{n+1}.$$

- Incremental form: $u^{n+1} - u^n$ has homogeneous boundary conditions (unless boundary conditions are time-dependent).
- For the time stepping method: set up linear combinations of matrices.
- (Only) if the sparsity pattern of the matrices agree we can copy the pattern and sum up the entries.

First, we create a matrix of the same size and sparsity pattern as m.mat:

In [ ]:

```
mstar = m.mat.CreateMatrix()
print(m.mat.nze, a.mat.nze, mstar.nze)
print(mstar)
```

To access the entries we use the vector of nonzero-entries:

In [ ]:

```
print(mstar.nze)
print(len(mstar.AsVector()))
```

In [ ]:

```
print(mstar.AsVector())
```

Using the vector we can build the linear combination of the a and the m matrix:

In [ ]:

```
mstar = m.mat.CreateMatrix()
mstar.AsVector().data = m.mat.AsVector() + dt * a.mat.AsVector()
invmstar = mstar.Inverse(freedofs=fes.FreeDofs())
```

We set the r.h.s. $f = exp(-6((x+\frac{1}{2})^2 + y^2)) - exp(-6((x-\frac{1}{2})^2 + y^2))$

In [ ]:

```
f = LinearForm(fes)
gaussp = exp(-6*((x+0.5)*(x+0.5)+y*y))-exp(-6*((x-0.5)*(x-0.5)+y*y))
Draw(gaussp,mesh,"f")
f += gaussp*v*dx
f.Assemble()
```

and the initial data: $u_0 = (1-y^2)x$

In [ ]:

```
gfu = GridFunction(fes)
gfu.Set((1-y*y)*x)
Draw(gfu,mesh,"u")
```

In [ ]:
```
res = gfu.vec.CreateVector()
tstep = 1 # time that we want to step over within one block-run
```

In [ ]:
```
%%time
t_intermediate=0 # time counter within one block-run
while t_intermediate < tstep - 0.5 * dt:
    res.data = dt * f.vec - dt * a.mat * gfu.vec
    gfu.vec.data += invmstar * res
    t_intermediate += dt
    print("\r",time+t_intermediate,end="")
    Redraw(blocking=False)
print("")
time+=t_intermediate
```

Alternative version with iterative solvers

- For a factorization of M^* (M^{*-1}) we require a sparse matrix M^*
- To store M^* as a sparse matrix requires new storage (and same memory layout of A and M)
- For iterative solvers we only require the matrix (and preconditioner) applications
- `mstar = m.mat + dt * a.mat` has no storage but matrix-vector multiplications

iterative solver version (with Jacobi preconditining):

In [ ]:
```
mstar_alt = m.mat + dt * a.mat
premstar_alt = m.mat.CreateSmoother() # + dt * a.mat.CreateSmoother()
from ngsolve.krylovspace import CGSolver
invmstar_alt = CGSolver(mstar_alt,premstar_alt, printrates='\r',\
                        tol=1e-8, maxiter=200)

print(premstar_alt)
```

In [ ]:
```
%%time
tstep = 0.1
t_intermediate=0 # time counter within one block-run
while t_intermediate < tstep - 0.5 * dt:
    res.data = dt * f.vec - dt * a.mat * gfu.vec
    gfu.vec.data += invmstar_alt * res
    t_intermediate += dt
    # print("\r t = {:24}, iteration steps: {}".format(time+t_intermediate,
    Redraw(blocking=False)
print("")
time+=t_intermediate
```

Supplementary material:

- Time dependent r.h.s.
- Time dependent boundary data
- Runge-Kutta time integration
- VTK Output

Supplementary 1: time-dependent r.h.s. data

Next: time-dependent r.h.s. data $f = f(t)$:

- Use `Parameter` t representing the time.
- A `Parameter` is a constant `CoefficientFunction` the value of which can be changed with the `Set` -function.

In [ ]:
```
t = Parameter(0.0)
```

An example of a time-dependent coefficient that we want to use as r.h.s. in the following is

In [ ]:
```
omega=1
gausspt = exp(-6*((x+sin(omega*t))*(x+sin(omega*t))+y*y))-exp(-6*((x-sin(om
scene = Draw(gausspt,mesh,"ft",order=3)
time = 0.0
from time import sleep
while time < 10 - 0.5 * dt:
    t.Set(time)
    scene.Redraw()
    time += 1e-3
    sleep(1e-3)
```

Accordingly we define a different linear form which then has to be assembled in every time step.

In [ ]:
```
ft = LinearForm(fes)
ft += gausspt*v*dx
time = 0.0
t.Set(0.0)
gfu.Set((1-y*y)*x)
#gfu.Set(CoefficientFunction(0))
scene = Draw(gfu,mesh,"u")
```

In [ ]:
```
tstep = 10 # time that we want to step over within one block-run
t_intermediate=0 # time counter within one block-run
res = gfu.vec.CreateVector()
while t_intermediate < tstep - 0.5 * dt:
    t.Set(time+t_intermediate+dt)
    ft.Assemble()
    res.data = dt * ft.vec - dt * a.mat * gfu.vec
    gfu.vec.data += invmstar * res
    t_intermediate += dt
    print("\r",time+t_intermediate,end="")
    scene.Redraw()
print("")
time+=t_intermediate
```

Supplementary 2: Time dependent boundary conditions

- $u|_{\partial\Omega} = u_D(t)$, $f = 0$
- implicit Euler time stepping method, non-incremental form:

$$M^* u^{n+1} = (M + \Delta t A) u^{n+1} = M u^n$$

- Homogenize w.r.t. to boundary conditions, i.e. we split

$$u^{n+1} = u_0^{n+1} + u_D^{n+1}$$

where u_D^{n+1} is a (discrete) function with correct boundary condition:

$$M^* u_0^{n+1} = M u^n - M^* u_D^{n+1}$$

In [ ]:

```
uD = CoefficientFunction( [(1-x*x)*IfPos(sin(0.3*pi*t),sin(0.3*pi*t),0),0,0
time = 0.0
t.Set(0.0)
gfu.Set(uD,BND)
scene = Draw(gfu,mesh,"u")
# visualization stuff
from ngsolve.internal import *
visoptions.mminval = 0.0
visoptions.mmaxval = 0.2
visoptions.deformation = 0
visoptions.autoscale = 0
```

In [ ]:

```
tstep = 2 # time that we want to step over within one block-run
t_intermediate=0 # time counter within one block-run
res = gfu.vec.CreateVector()
while t_intermediate < tstep - 0.5 * dt:
    t.Set(time+t_intermediate+dt)
    res.data = m.mat * gfu.vec
    gfu.Set(uD,BND)
    res.data -= mstar * gfu.vec
    gfu.vec.data += invmstar * res
    t_intermediate += dt
    print("\r",time+t_intermediate,end="")
    scene.Redraw()
print("")
time+=t_intermediate
```

Supplementary 3: Singly diagonally implicit Runge-Kutta methods

We consider more sophisticated time integration methods, SDIRK methods. To simplify presentation we set $f = 0$.

SDIRK methods for the semi-discrete problem $\frac{d}{dt} u = M^{-1} F(u) = -M^{-1} \cdot Au$ are of the form:

$$u^{n+1} = u^n + \Delta t M^{-1} \sum_{i=0}^{s-1} b_i k_i$$

with

$$k_i = -A u_i \qquad \text{where } u_i \text{ is the solution to} \qquad (M + a_{ii} \Delta t A) u_i = M u^n - \Delta t \sum_{i=0}^{i-1} a_{ij} k_j,$$

The coefficients a,b and c are stored in the butcher tableau:

$$\begin{array}{c|ccccc} c_0 & a_{00} & 0 & \ddots & & \\ c_1 & a_{10} & a_{11} & 0 & \ddots & \\ \vdots & \vdots & \vdots & \ddots & 0 & \\ c_{s-1} & a_{s-1,0} & a_{s-1,1} & \dots & a_{s-1,s-1} \\ \hline & b_0 & b_1 & \dots & b_{s-1} \end{array}$$

For an SDIRK method we have $a^* = a_{ii},\ i = 0, .., s-1$.

Simplest example: Implicit Euler

$$\begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array}$$

In [ ]:

```
class sdirk1: #order 1 (implicit Euler)
    stages = 1
    a = [[1]]
    b = [1]
    c = [1]
    astar = 1
```

We can use for example the 2 stage SDIRK (order 3) method

$$\begin{array}{c|cc} p & p & 0 \\ 1-p & 1-2p & p \\ \hline & 1/2 & 1/2 \end{array}$$

with $p = \frac{3-\sqrt{3}}{6}$.

In [ ]:

```
class sdirk2: #order 3
    p = (3 - sqrt(3))/6
    stages = 2
    a = [[p, 0],
         [1 - 2*p, p]]
    b = [1/2, 1/2]
    c = [p, 1 - p]
    astar = p
```

We can use for example the 5 stage SDIRK (order 4) method

$$\begin{array}{c|ccccc} 1/4 & 1/4 & & & & \\ 3/4 & 1/2 & 1/4 & & & \\ 11/20 & 17/50 & -1/25 & 1/4 & & \\ 1/2 & 371/1360 & -137/2720 & 15/544 & 1/4 & \\ 1 & 25/4 & -49/48 & 125/16 & -85/12 & 1/4 \\ \hline & 25/4 & -49/48 & 125/16 & -85/12 & 1/4 \end{array}$$

In [ ]:
```
class sdirk5: #order 4
    stages = 5
    a = [[1/4, 0, 0, 0, 0],
         [1/2, 1/4, 0, 0, 0],
         [17/50,-1/25, 1/4, 0, 0],
         [371/1360, -137/2720, 15/544, 1/4,0],
         [25/24, -49/48, 125/16, -85/12, 1/4]]
    b = [25/24, -49/48, 125/16, -85/12, 1/4]
    c = [1/4, 3/4, 11/20, 1/2, 1]
    astar = 1/4
```

SDIRK2 :

In [ ]:
```
butchertab = sdirk2()
rhsi = gfu.vec.CreateVector()
Mu0 = gfu.vec.CreateVector()
ui = gfu.vec.CreateVector()
k = [gfu.vec.CreateVector() for i in range(butchertab.stages)]
```

We have to update the M^* matrix and reset initial data

In [ ]:
```
time = 0.0
t.Set(0.0)
gfu.Set(uD,BND)
scene = Draw(gfu,mesh,"u")
# visualization stuff
from ngsolve.internal import *
visoptions.mminval = 0.0
visoptions.mmaxval = 0.2
visoptions.deformation = 0
visoptions.autoscale = 0

mstar.AsVector().data = m.mat.AsVector() + butchertab.astar * dt * a.mat.As
invmstar = mstar.Inverse(freedofs=fes.FreeDofs())
invmass = m.mat.Inverse(freedofs=fes.FreeDofs())
```

In [ ]:
```
tstep = 2 # time that we want to step over within one block-run
t_intermediate=0 # time counter within one block-run
while t_intermediate < tstep - 0.5 * dt:
    Mu0.data = m.mat * gfu.vec
    for i in range(butchertab.stages):
        # add up the ks as prescribed in the butcher tableau
        rhsi.data = Mu0
        for j in range(0,i):
            rhsi.data += dt * butchertab.a[i][j] * k[j]
        # Solve for ui (with homogenization for the boundary data)
        t.Set(time+t_intermediate+butchertab.c[i]*dt)
        gfu.Set(uD,BND)
        ui.data = gfu.vec; rhsi.data -= mstar * ui
        ui.data += invmstar * rhsi
        # compute k[i] from ui
        k[i].data = - a.mat * ui
    t_intermediate += dt; t.Set(time+t_intermediate)
    # Adding up the ks:
    gfu.Set(uD,BND)
    Mu0.data -= m.mat * gfu.vec
    for i in range(0,butchertab.stages):
        Mu0.data += dt * butchertab.b[i] * k[i]
    gfu.vec.data += invmass * Mu0
    print("\r",time+t_intermediate,end="")
```

```
    scene.Redraw()
print(""); time+=t_intermediate
```

Supplementary 4: VTK Output ("exporting the nice pictures")

- see also https://ngsolve.org/blog/ngsolve/2-vtk-output
- or `py_tutorials/vtkout.py` (ngsolve repository)

Outputting the nice pictures to vtk (to visualize with paraview):

In [ ]:
```
vtk = VTKOutput(mesh,coefs=[gfu],
                names=["sol"],
                filename="vtk_example1",
                subdivision=3)
vtk.Do()
```

In [ ]:
```
%%bash
ls vtk_example1.*
```

You can also export vector fields:

In [ ]:
```
vtk = VTKOutput(mesh,coefs=[gfu,grad(gfu)],names=["sol","gradsol"],filename
vtk.Do()
```

In [ ]:
```
#%%bash
#paraview vtk_example2.vtk
```

And time dependent data:

In [ ]:
```
vtk = VTKOutput(mesh,coefs=[gfu],names=["sol"],filename="vtk_example3",subd
gfu.Set((1-y*y)*x)
vtk.Do(time=0)
time = 0
tstep = 1 # time that we want to step over within one block-run
t_intermediate=0 # time counter within one block-run
res = gfu.vec.CreateVector()
i = 0
while t_intermediate < tstep - 0.5 * dt:
    res.data = dt * f.vec - dt * a.mat * gfu.vec
    gfu.vec.data += invmstar * res
    t_intermediate += dt
    print("\r",time+t_intermediate,end="")
    scene.Redraw()
    i += 1
    if (i%10 == 0):
        vtk.Do(time=t_intermediate)
print("")
```

Call paraview on gerenated data (if installed):

In [ ]:
```
%%bash
if ! command -v paraview &> /dev/null
then
```

```
    echo "paraview is not installed."
    exit
else
    paraview vtk_example3.pvd
fi
```

In [ ]: