Stationary Navier-Stokes

In [ ]:
```
using Revise
using CairoMakie
using Symbolics
using Latexify
using BenchmarkTools
include("RBFunctions.jl")
using Plots
using LinearAlgebra
```

In [ ]:

```
@variables ϵ r x₁ x₂ t Δt;
const nu = 1.0
μ = 0.02
ρ = 1.0
#ϕ = 1//945 * ((ϵ*r)^5 +15*(ϵ*r)^3 + 105*(ϵ*r)^2 + 945*(ϵ*r)+ 945)* exp(-ϵ*
r)
φ = exp(-r^2*ϵ^2)
φ = substitute(φ, r=>sqrt(x₁^2 + x₂^2))
#display(ϕ)
Δ(exprs) = expand_derivatives((Differential(x₁)^2)(exprs) + (Differential(x
₂)^2)(exprs))
∂₁(exprs) = expand_derivatives(Differential(x₁)(exprs))
∂₂(exprs) = expand_derivatives(Differential(x₂)(exprs))
∂ₜ(exprs) = expand_derivatives(Differential(t)(exprs))

Φ_div = ([-∂₂(∂₂(φ)) ∂₁(∂₂(φ)) 0.0 ; ∂₁(∂₂(φ)) -∂₁(∂₁(φ)) 0.0; 0.0 0.0 φ])
ΔΦ_div= Δ.([-∂₂(∂₂(φ)) ∂₁(∂₂(φ)); ∂₁(∂₂(φ)) -∂₁(∂₁(φ))])
Φ_curl = ([-∂₁(∂₁(φ)) -∂₁(∂₂(φ)); -∂₁(∂₂(φ)) -∂₂(∂₂(φ))])
Φ = [φ 0.0 0.0; 0.0 φ 0.0;0.0 0.0 φ]
#ΔΦ = [Δ(ϕ) 0 ; 0 Δ(ϕ)]

f₁ = 0.0
f₂ = 0.0
f₁ = eval(build_function(f₁,x₁, x₂, t))
f₂ = eval(build_function(f₂,x₁, x₂, t))
zero_func(x₁,x₂,t) = 0.0


λ1y(x) = (μ/ρ)*Δ(x[1]) +  (1/ρ)*∂₁(x[3])
λ2y(x) = (μ/ρ)*Δ(x[2]) +  (1/ρ)*∂₂(x[3])
λ3y(x) = x[1]
λ4y(x) = x[2]

λ1x(x) = (μ/ρ)*Δ(x[1]) - (1/ρ)*∂₁(x[3])
λ2x(x) = (μ/ρ)*Δ(x[2]) -  (1/ρ)*∂₂(x[3])
λ3x(x) = x[1]
λ4x(x) = x[2]

λu(x) = x[1]
λv(x) = x[2]
λp(x) = x[3]

λ∂₁u(x) = ∂₁(x[1])
λ∂₂u(x) = ∂₂(x[1])
λ∂₁v(x) = ∂₁(x[2])
λ∂₂v(x) = ∂₂(x[2])
```

Out[ ]:

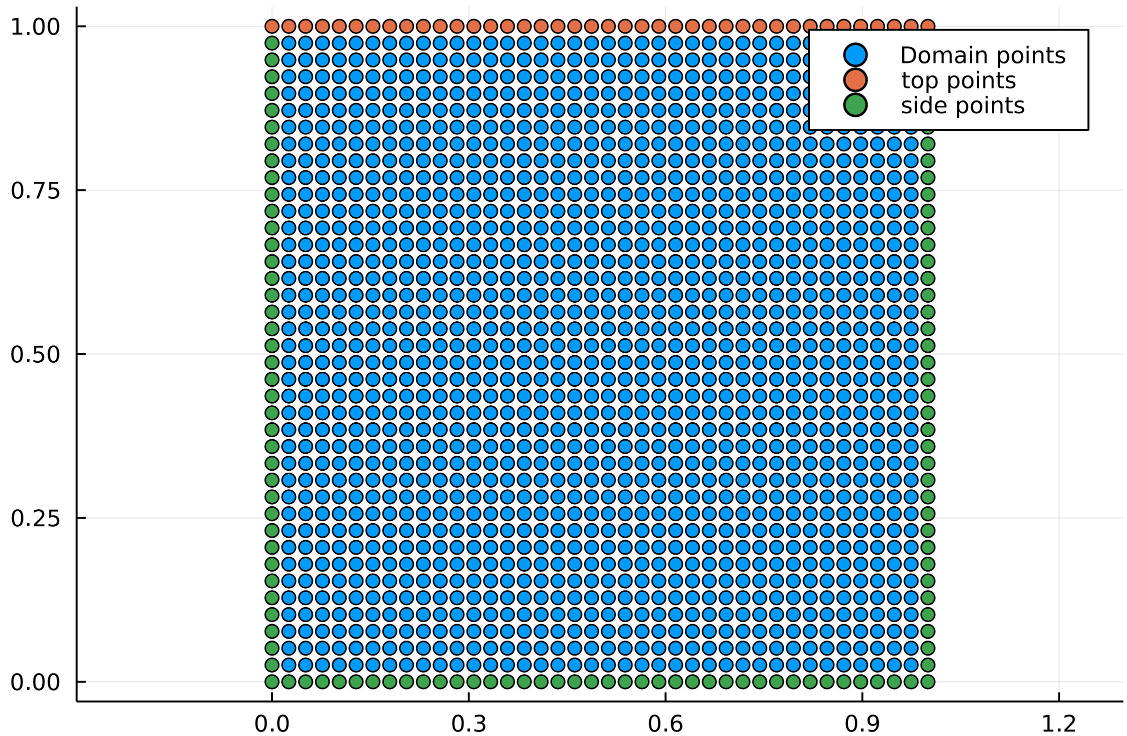
```
λ∂₂v (generic function with 1 method)
```

In [ ]:
```
#generate points for lid_driven_cavity
Internal_points,Boundary_points =  generate_2D_equally_spaced_points(40)
N_i = size(Internal_points)[2]
N_b = size(Boundary_points)[2]
N = N_i + N_b
top_points= zeros((2,1+Int(N_b/4)))
side_points = zeros((2,N_b-size(top_points)[2]))
s1,s2 = 1,1
for i in 1:N_b
    if Boundary_points[2,i] == 1.0
        top_points[:,s1] = Boundary_points[:,i]
        s1+=1
    else
        side_points[:,s2] = Boundary_points[:,i]
        s2+=1
    end
end
Boundary_points = hcat(top_points,side_points)
All_points = hcat(Internal_points,Boundary_points)

N_top = size(top_points)[2]
N_side = size(side_points)[2]
println("total number of nodes: ",N)
println("number of internal_nodes: ",N_i)
println(" number of top nodes: ",N_top)
println("number of side nodes: ",N_side)
```

```
total number of nodes: 1600
number of internal_nodes: 1444
 number of top nodes: 40
number of side nodes: 116
```

In [ ]:
```
Plots.scatter(Internal_points[1,:],Internal_points[2,:],aspect_ratio=:equa
l,label = "Domain points")
Plots.scatter!(top_points[1,:],top_points[2,:],aspect_ratio=:equal,label ="
top points")
Plots.scatter!(side_points[1,:],side_points[2,:],aspect_ratio=:equal,label
="side points")
#savefig("internal_domain_points.png")
```

Out[ ]:



In [ ]:
```
Eval_points, _ =  generate_2D_equally_spaced_points(50)
N_eval = size(Eval_points)[2]
```

Out[ ]: 2304

Construct matrices

In [ ]:
```
parameter = 16

# Stokes matrix
A_functions = construct_kernel_array(Φ_div,[λ1x,λ2x,λ3x,λ4x],[λ1y,λ2y,λ3y,λ
4y])
A_functions= compile_kernel_array(A_functions)
A_tensor = crete_block_point_tensors([Internal_points,Internal_points,Bound
ary_points,Boundary_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
A = generate_block_matrices(A_functions,A_tensor,parameter)
A = flatten(A)

# transformation of coefficients to velocities and pressure
E_functions = construct_kernel_array(Φ_div,[λu,λv,λp],[λ1y,λ2y,λ3y,λ4y])
E_functions = compile_kernel_array(E_functions)
E_tensor = crete_block_point_tensors([Eval_points,Eval_points,Eval_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
E = generate_block_matrices(E_functions,E_tensor,parameter)
E = flatten(E)

# Matrices for nonlinear terms

U_functions = construct_kernel_array(Φ_div,[λu],[λ1y,λ2y,λ3y,λ4y])
#display(U_functions)
U_functions = compile_kernel_array(U_functions)
U_tensor = crete_block_point_tensors([Internal_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
U = generate_block_matrices(U_functions,U_tensor,parameter)
U = flatten(U)

V_functions = construct_kernel_array(Φ_div,[λv],[λ1y,λ2y,λ3y,λ4y])
V_functions = compile_kernel_array(V_functions)
V_tensor = crete_block_point_tensors([Internal_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
V = generate_block_matrices(V_functions,V_tensor,parameter)
V = flatten(V)

Ux_functions = construct_kernel_array(Φ_div,[λ∂₁u],[λ1y,λ2y,λ3y,λ4y])
Ux_functions = compile_kernel_array(Ux_functions)
Ux_tensor = crete_block_point_tensors([Internal_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
Ux = generate_block_matrices(Ux_functions,Ux_tensor,parameter)
Ux = flatten(Ux)

Uy_functions = construct_kernel_array(Φ_div,[λ∂₂u],[λ1y,λ2y,λ3y,λ4y])
Uy_functions = compile_kernel_array(Uy_functions)
Uy_tensor = crete_block_point_tensors([Internal_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
Uy = generate_block_matrices(Uy_functions,Uy_tensor,parameter)
Uy = flatten(Uy)


Vx_functions = construct_kernel_array(Φ_div,[λ∂₁v],[λ1y,λ2y,λ3y,λ4y])
Vx_functions = compile_kernel_array(Vx_functions)
Vx_tensor = crete_block_point_tensors([Internal_points],
```

```
[Internal_points,Internal_points,Boundary_points,Boundary_points])
Vx = generate_block_matrices(Vx_functions,Vx_tensor,parameter)
Vx = flatten(Vx)

Vy_functions = construct_kernel_array(Φ_div,[λ∂₂v],[λ1y,λ2y,λ3y,λ4y])
Vy_functions = compile_kernel_array(Vy_functions)
Vy_tensor = crete_block_point_tensors([Internal_points],
[Internal_points,Internal_points,Boundary_points,Boundary_points])
Vy = generate_block_matrices(Vy_functions,Vy_tensor,parameter)
Vy = flatten(Vy)


println(cond(A))


function g(t)
    res = zeros(N_b*2)
    res[1:N_top] .= min(t,8.0)
    return res
end

f = generate_vector_function([f₁,f₂],Internal_points)

g(1.2)

matrices = [A,U,Ux,Uy,V,Vx,Vy]
print("done")
```

```
4.862732383415179e10
done
```

In [ ]:

```
function assemble_matrix!(M,c,mat)
    A,U,Ux,Uy,V,Vx,Vy = mat
    N = size(A)[1]
    N_i = size(U)[1]
    N_b = N - N_i
    u = U*c
    v = V*c
    M[1:N_i,1:N] .= (u .* Ux) .+ (v .* Uy)
    M[1+N_i:2*N_i,1:N] .= (u .* Vx) .+ (v .* Vy)
    M .= M .+ A
    return M

end
function assemble_RHS!(RHS,c,mat)
    A,U,Ux,Uy,V,Vx,Vy = mat
    N = size(A)[1]
    N_i = size(U)[1]
    N_b = N - N_i
    u = U*c
    v = V*c
    RHS[1:N_i] .= (u .* (Ux*c)) .+ (v .* (Uy*c))
    RHS[1+N_i:2*N_i] .= (u .* (Vx*c)) .+ (v .* (Vy*c))
end
```

Out[ ]:

```
assemble_RHS! (generic function with 1 method)
```

Solve using Picard Linearization for Re = 50

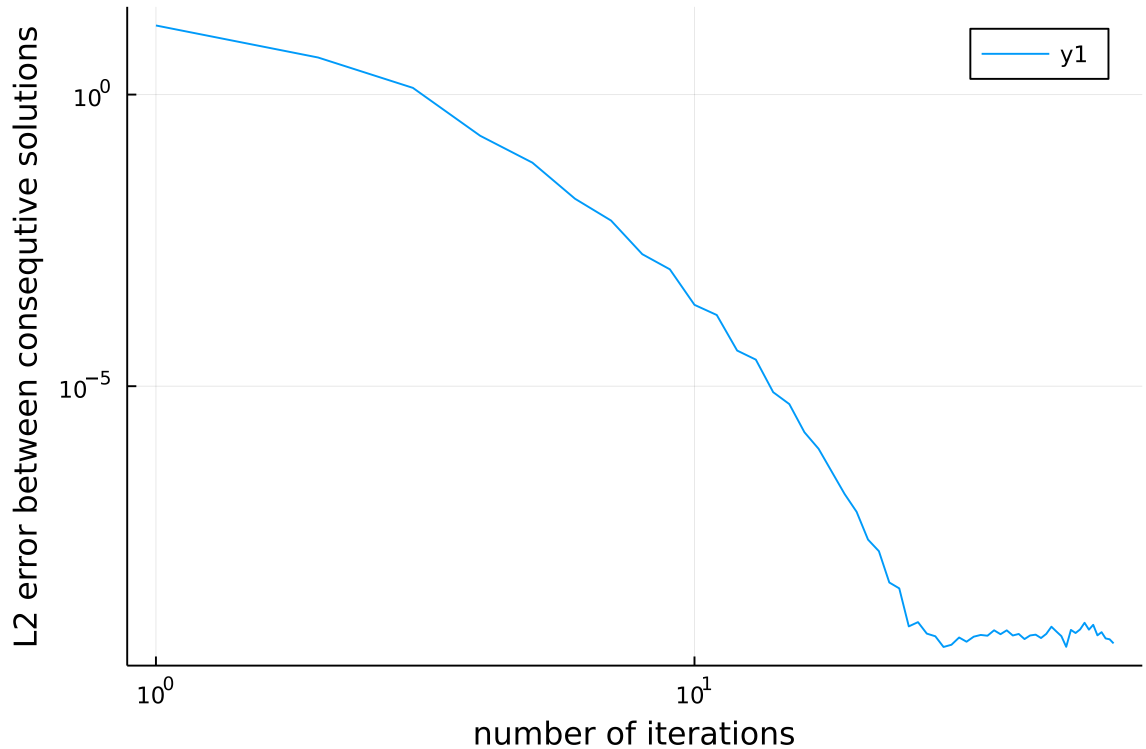
In [ ]:
```
RHS = zeros(2*N)
RHS[2*N_i+1:2*N_i+N_top] .= ones(N_top)

c = zeros(2*N)
c_old = zeros(2*N)
M = zeros((2*N,2*N))
N_iter = 60
error_array = []
for i in 1:N_iter
    assemble_RHS!(RHS,c,matrices)
    c_old .= c
    c = A\RHS
    append!(error_array,[norm(c-c_old)])
end
#c = A\RHS
sol = E*c # calculate u,v and p at evaluation points
```

Out[ ]:
```
6912-element Vector{Float64}:
 -0.006940726790612074
 -0.0007887000612648522
 -0.002312157972463257
 -0.001172717732060517
 -0.001226940302482752
 -0.0016384863485134381
 -0.001126536971920617
 -0.0017259942304983927
 -0.0015304280582590115
 -0.0016855864831533279
 -0.0019702656795481813
 -0.0018174671921481846
 -0.0022153868009896003
  ⋮
  0.1284334140593818
  0.14979778131650692
  0.17460989428379706
  0.20372840202917458
  0.23564381913609023
  0.27165366006141894
  0.3136716104046529
  0.3575321107862197
  0.403984034360903
  0.46191279167520893
  0.5177399636917421
  0.5243558844579905
```

In [ ]:

```
Plots.plot(1:N_iter,error_array,yscale = :log10,xscale = :log10,xlabel="num
ber of iterations",
ylabel = "L2 error between consequtive solutions")
#error_array
#println(maximum(c))
#println(maximum(RHS))
```

Out[ ]:



In [ ]:

```
maximum(sol)
```

Out[ ]: 0.7682810846365004

In [ ]:

```
fig = Figure(resolution = (800, 800))
strength = sqrt.(sol[1:N_i] .^2 .+ sol[1+N_i:2*N_i] .^2)
println(minimum(strength))
Axis(fig[1, 1], backgroundcolor = "black")
arrows!(Eval_points[1,:], Eval_points[2,:], sol[1:N_eval], sol[1+N_eval:2*N
_eval], arrowsize = 5,lengthscale = 0.15,
arrowcolor = :white, linecolor = :white)
fig
```

```
0.004868766428909093
```

Out[ ]: