```
begin
    using PlutoUI
    using MAT
    using LinearAlgebra, Statistics
    using Plots
end
```

Project : Reduced-order modeling of the two-dimensional cylinder flow

The two-dimensional cylinder flow is a canonical of bluff body flows in fluid dynamics. Despite its simplicity, this two-dimensional flow captures some fundamental features of larger scale flows as shown in the image below.

This flow pattern is known as the **Bénard-von Kàrmàn vortex street** and corresponds to the periodic shedding of vortices from the cylinder (or the island in this image above).

From a mathematical point of view, the dynamics of this flow can be modeled by the incompressible Navier-Stokes equations

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u} \otimes \mathbf{u}) = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u}$$
$$\nabla \cdot \mathbf{u} = 0$$

where $\mathbf{u}$ is the two-dimensional velocity field, p the pressure field and Re the Reynolds number (i.e. the ratio of the inertial and viscous forces). In order to simulate these equations on a computer, they are typically discretized on a mesh with n points such that $\mathbf{u} \in \mathbb{R}^{2n}$ and $p \in \mathbb{R}^n$ where n can be of the order of several hundred thousands. From a dynamical point of view however, the dynamics are rather simple and correspond to simple periodic dynamics. As such, dynamically, only two degrees of freedom (e.g. the amplitude and the phase of the oscillation) are needed to characterize the state of the system rather than n degrees. In this notebook, you will have to analyze these dynamics using the different tools discussed during the class (i.e. dimensionality reduction using PCA or DMD, sparse sensor placement, least-squares and compressive sensing). For that purpose, you'll be given a dataset consisting in snapshots of the vorticity field (i.e. $\omega = \nabla \times \mathbf{u}$) over time obtained by direct numerical simulation of the Navier-Stokes equations. A random selection of these snapshots is shown below.

Dataset to be loaded :

```
begin
    # --> Load the dataset.
    data = matread("cylinder_dataset.mat")

    # --> Extract the mesh and the snapshots collection.
    mesh, X = [data["x"][:], data["y"][:]], data["data"]
end;
```

Display random snapshots from the database : Click

Dimensionality reduction

As a starting point, let us try to determine the intrinsic dimensionality of our dataset. For that purpose, we'll use *principal component analysis* (PCA). A brief recap' of PCA is given below.

Principal Component Analysis

PCA is closely related to the SVD matrix factorization. In this notebook, we will however use PCA through its correlation matrix formulation. Given a data matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ where each column

corresponds to one snapshot of the cylinder flow simulation with m state variables and we have n such snapshots sampled uniformly in time, let us first compute the mean flow, i.e.

$$\overline{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i.$$

In a second step, the temporal correlation matrix

$$\mathbf{\Sigma_X} = \frac{1}{n-1}\sum_{i=1}^{n} (\mathbf{x}_i - \overline{\mathbf{x}})^T(\mathbf{x}_i - \overline{\mathbf{x}})$$

is constructed. Assuming that the mean flow has already been subtracted from the columns of $\mathbf{X}$ (`X .-= x̄` in Julia), this correlation matrix can easily be computed in one go as follows

$$\mathbf{\Sigma_X} = \frac{1}{n-1}\mathbf{X}^T\mathbf{X}.$$

In a third step, the eigendecomposition of $\mathbf{\Sigma_X}$ is computed, i.e.

$$\mathbf{\Sigma_X V} = \mathbf{V\Lambda}$$

where $\mathbf{\Lambda}$ is the diagonal matrix of eigenvalues and $\mathbf{V}$ is the corresponding matrix of eigenvectors normalized such that $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. The PCA modes can then be computed as

$$\mathbf{U} = \frac{1}{\sqrt{N-1}}\mathbf{X V \Lambda}^{-\frac{1}{2}}$$

while the projection of the snapshots into this particular basis is given by

$$\mathbf{a} = \mathbf{U}^T\mathbf{X}.$$

Application to the cylinder flow

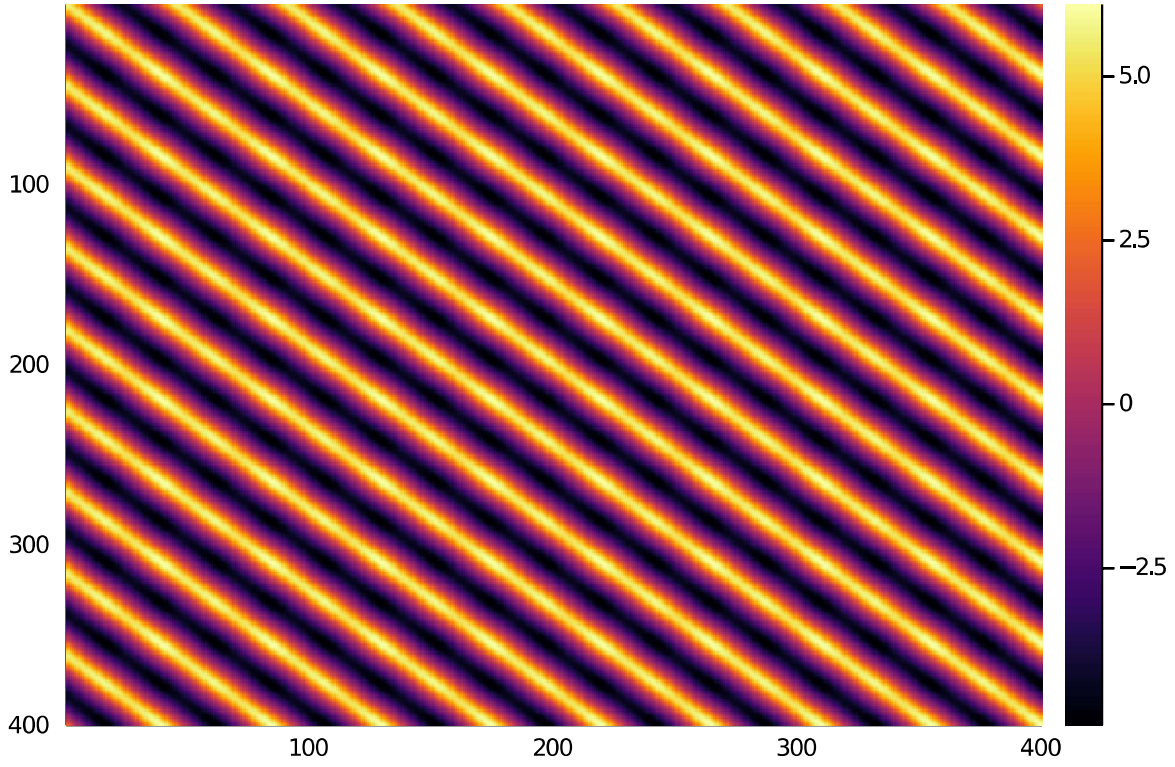
It is now up to you. Given the dataset already uploaded in the notebook as `X`, use the cells below to :

1. Compute the mean flow `x̄` and plot it using the `plot_flow_field(mesh, x̄)` command.
2. Compute the covariance matrix `Σ` and plot it using `heatmap(Σ)`. How do you interpret this plot?
3. Compute the eigendecomposition of `Σ` using `eigen(Σ)` and plot the distribution of the eigenvalues. Given that each eigenvalue is directly related to the amount of turbulent kinetic energy captured by the corresponding PCA mode, how many modes do you need to keep in your analysis to capture 99% of the kinetic energy ?
4. Compute and plot the first four PCA modes using `plot_flow_field(mesh, U[:, i])`. Eventhough you may not be familiar with fluid dynamics, try to explain the patterns you observe.

```
begin
# --> Compute the mean flow.
    m,n = size(X);
    #Xs = sum(X, dims=2)
    #Xm = Xs/n
    Xm = mean(X, dims=2)
    plot_flow_field(mesh, Xm)
end
```

```
1.4546697180151113e-15
```

```
maximum(Xm)
```



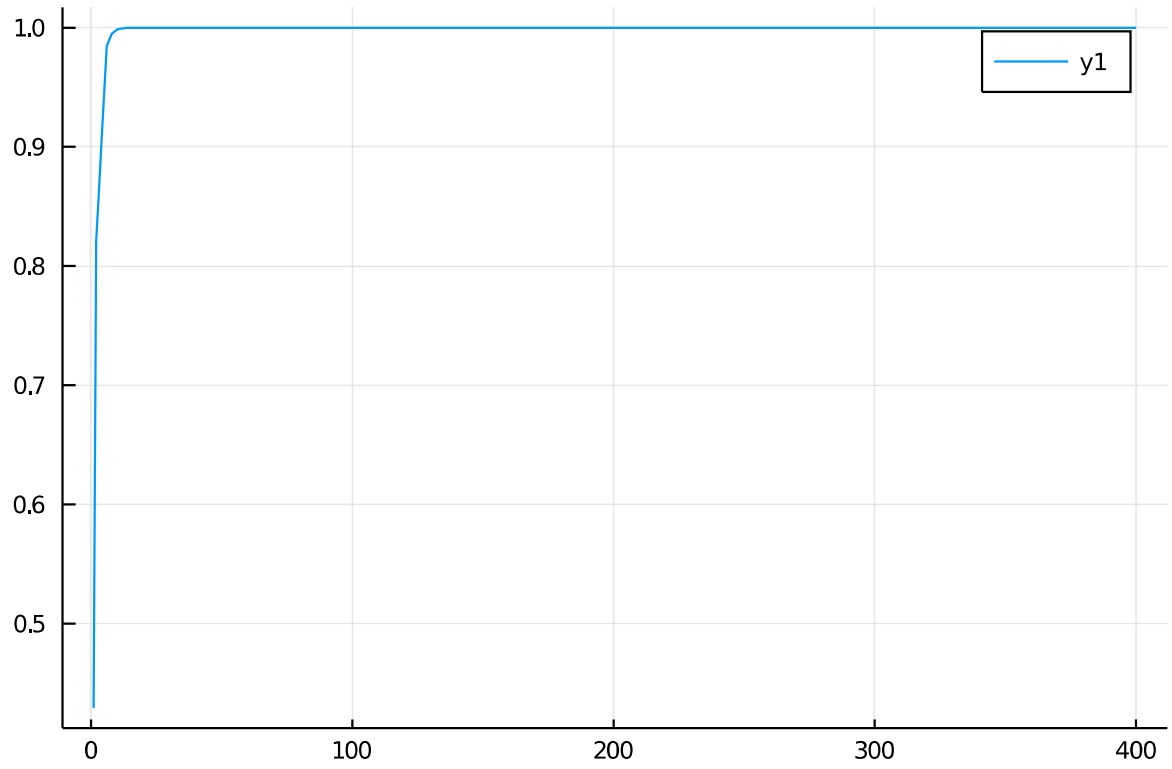
```
begin
    # --> Subtract mean from the data matrix.
    B = X.-Xm

    # --> Compute the covariance matrix.
    Σ = (1/(n-1)) * (B'*B)
    # --> Plot the covariance matrix.
    heatmap(Σ, yflip=true)

end
```

Interpration :

```
md"
 **Interpration :**
"#From the heatmap we can observe a very structured and repetitive structure.
Therefore we can infer that our system has some sort of periodic behaviour and tha
it is possible to reconstruct it in an accurate manner using only a few PCA modes.
```




```
begin
    # --> Compute the eigendecomposition of Σ.
    Λ, V = eigen(Σ) #Λ are the eigen values and V its eigenvectors
    # --> Plot the distribution of eigenvalues.
    Λ = Λ[end:-1:1]
    V = V[:,end:-1:1]
    plot(cumsum(Λ)./sum(Λ))

end
```

```
Λd =
400×400 Diagonal{Float64,Array{Float64,1}}:
 972.397     ⋅        ⋅        ⋅        ⋅        ⋅      …    ⋅          ⋅
   ⋅     886.729     ⋅        ⋅        ⋅        ⋅           ⋅          ⋅
   ⋅         ⋅     98.6818     ⋅        ⋅        ⋅           ⋅          ⋅
   ⋅         ⋅        ⋅      94.749     ⋅        ⋅           ⋅          ⋅
   ⋅         ⋅        ⋅         ⋅     89.6219     ⋅           ⋅          ⋅
   ⋅         ⋅        ⋅         ⋅        ⋅      87.7482  …    ⋅          ⋅
   ⋅         ⋅        ⋅         ⋅        ⋅           ⋅          ⋅          ⋅
  ⋮                                              ⋮       ⋱
```

```
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮          ⋮          ⋮
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮     ⋱    ⋮          ⋮
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮          ⋮          ⋮
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮          ⋮          ⋮
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮    -1.34828e-13     ⋮
    ⋮        ⋮        ⋮        ⋮        ⋮        ⋮          ⋮     -2.55778e-13
```

```
Λd = Diagonal(Λ)
```

20

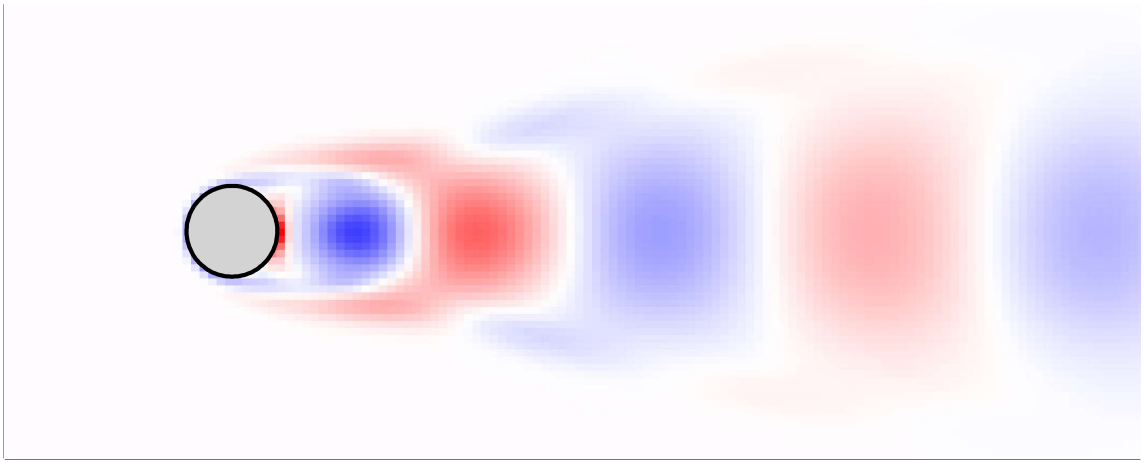
```
begin
    trg = 0.99999 * tr(Λd)  #Desired % of Kinetic energy to be captured
    i=1                     #Iteration counter
    val = Λ[1]              #First eigenvalue
    while val<=trg          #Checks if current modes capture the desired energy
        val = val + Λ[i+1] #If not, add energy or the next eigenvalue
        i = i+1             #Increase the iteration counter
    end
    modes = i            #Number of PCA modes to be considered
end
```

100

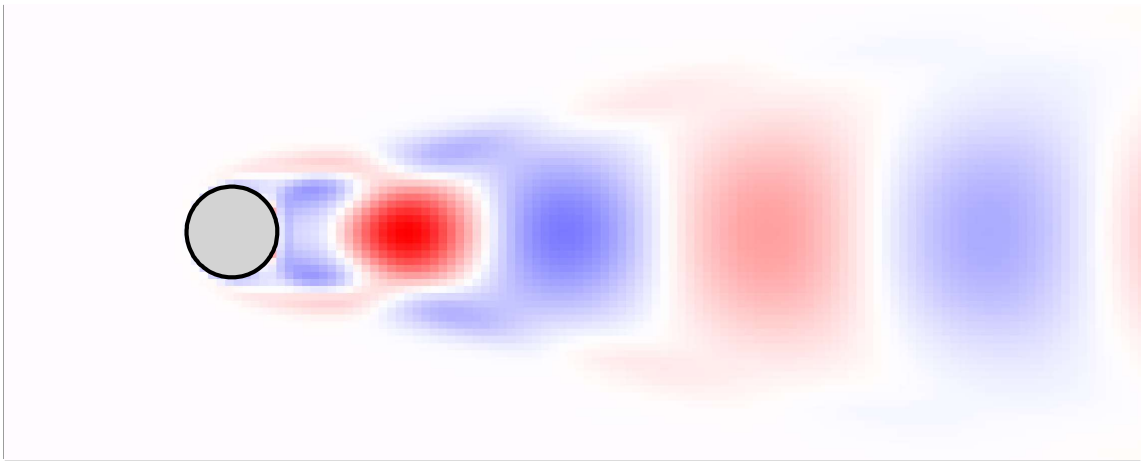
```
begin
    nmodes = modes + 80 #Adding extra modes for comparision purposes.
end
```

```
U = 20769×100 Array{Float64,2}:
 -3.46018e-6   1.37692e-6  -4.09182e-8   2.16315e-8  …   6.92496e-5    2.68008e-5
 -3.47626e-6   1.3893e-6   -4.03592e-8   2.06432e-8      2.25224e-5    3.04538e-5
 -3.49132e-6   1.40255e-6  -3.97073e-8   1.97293e-8      5.44934e-6    1.87881e-5
 -3.50761e-6   1.4138e-6   -3.9224e-8    1.87091e-8      4.37491e-5    4.99812e-5
 -3.52456e-6   1.424e-6    -3.88307e-8   1.7631e-8       0.00010568    0.000104202
 -3.53885e-6   1.43719e-6  -3.81906e-8   1.67658e-8  …   0.000121789   9.53808e-5
 -3.55466e-6   1.44816e-6  -3.77237e-8   1.57648e-8      5.48673e-5    4.8082e-5
  ⋮                                                ⋱
  1.56522e-6  -3.0167e-6   -1.78447e-7  -1.4641e-7       -2.61594e-5   -1.50173e-5
  1.55773e-6  -3.00776e-6  -1.58547e-7  -1.25877e-7      -4.43167e-5   -8.65964e-6
  1.54782e-6  -2.99865e-6  -1.4002e-7   -1.12861e-7  …   -6.37844e-5    3.82088e-5
  1.53776e-6  -2.98957e-6  -1.21434e-7  -1.00357e-7      -8.35703e-5    8.69876e-5
  1.52996e-6  -2.97817e-6  -1.09257e-7  -8.27941e-8      -0.000102134   0.000130714
  1.52208e-6  -2.96498e-6  -9.69583e-8  -7.47447e-8      -0.00011932    0.000148214
```

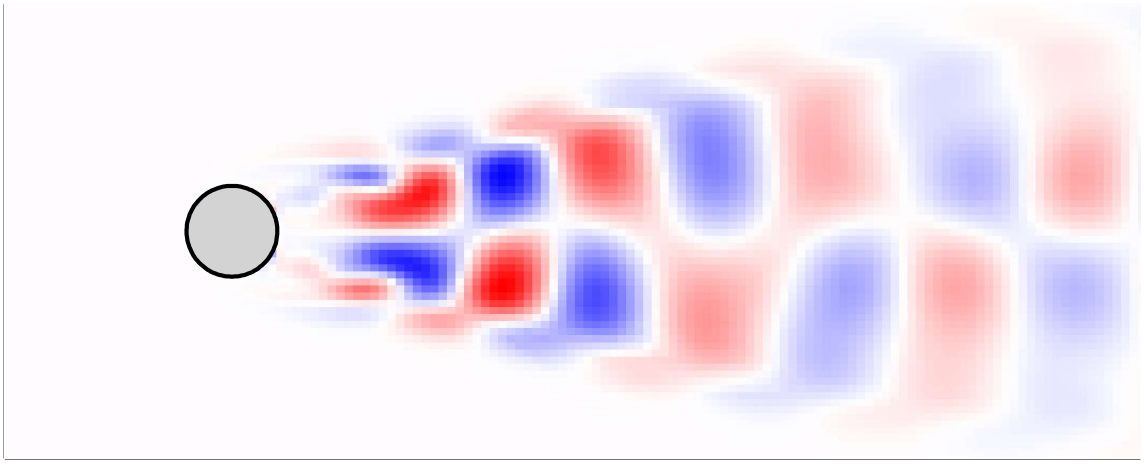
```
# --> Compute the PCA modes.
# Truncation to 99%
U = (1/sqrt(n-1))*B*V[:,1:nmodes]*Λd[1:nmodes,1:nmodes]^-0.5
```



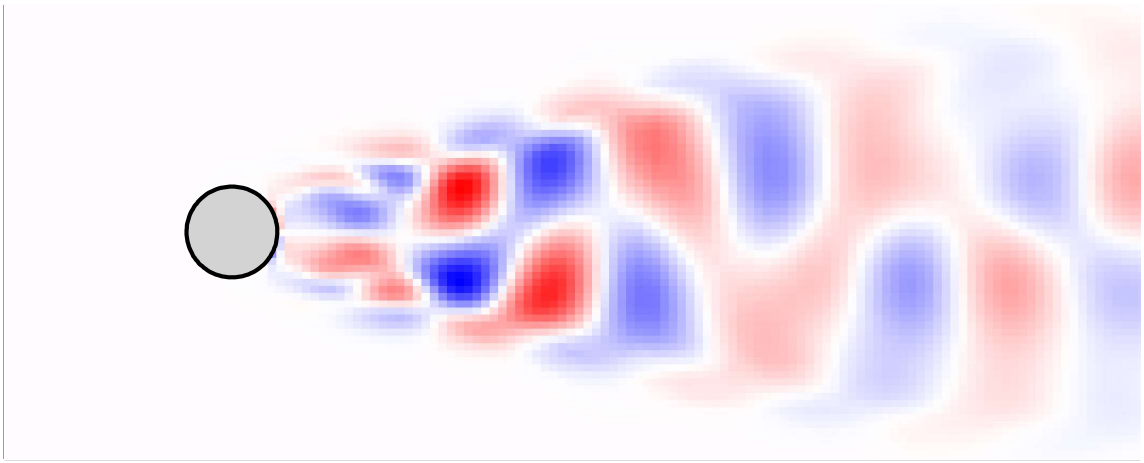
```
# --> Plot PCA 1.
plot_flow_field(mesh, U[:, 1])
```



```
# --> Plot PCA 2.
plot_flow_field(mesh, U[:, 2])
```

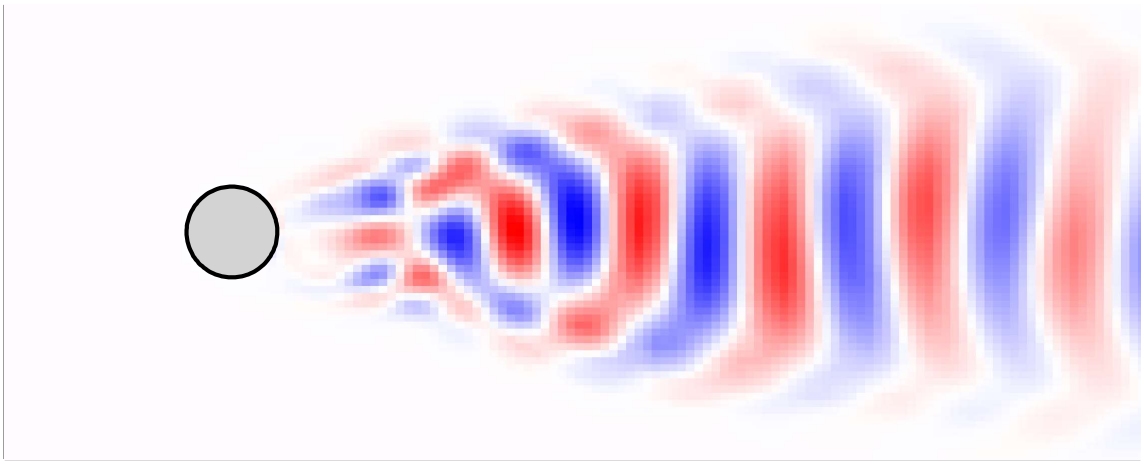


```
# --> Plot PCA 3.
plot_flow_field(mesh, U[:, 3])
```

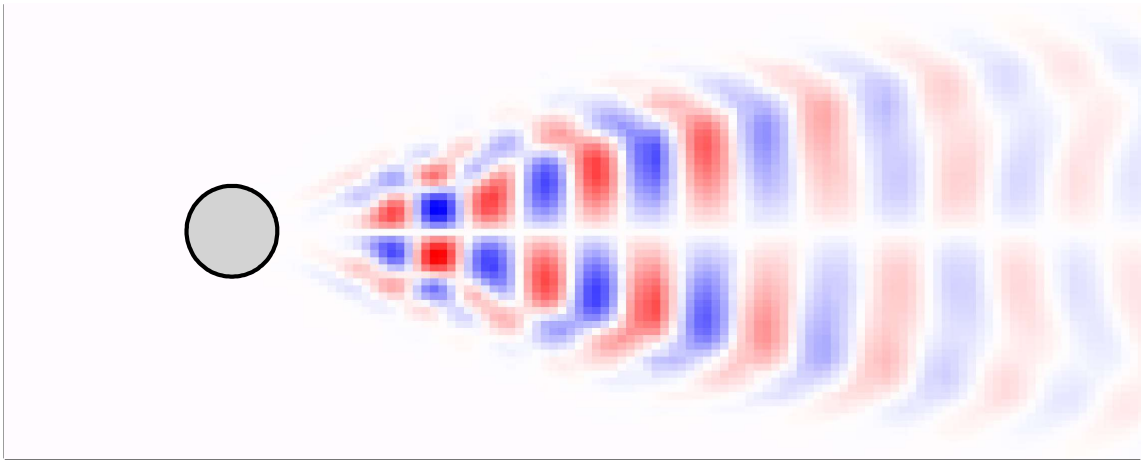


```
# --> Plot PCA 4.
plot_flow_field(mesh, U[:, 4])
```

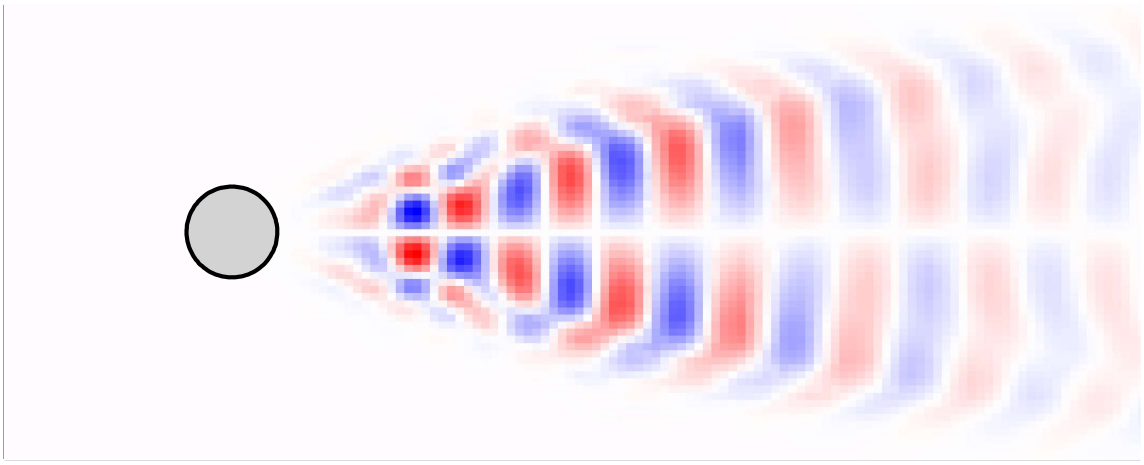
```
# --> Plot PCA 5.
plot_flow_field(mesh, U[:, 5])
```



```
# --> Plot PCA 6.
plot_flow_field(mesh, U[:, 6])
```



```
# --> Plot PCA 7.
plot_flow_field(mesh, U[:, 7])
```



```
# --> Plot PCA 8.
plot_flow_field(mesh, U[:, 8])
```

Pattern :

```
md"
**Pattern :**
"#According to our results, we would need at least 8 PCA modes to capture 99% of the
Kinetic energy.
```

```
#From my very basic understanding of Fluid mechanics, I believe these snapshots to be the behaviour of the vortexes formed when the fluid colides with the object over time. This would explain why with this 8 snapshots we can reconstruct the whole model by making linear combinations of them, since the model has a periodic behaviour that repeats itself each cycle. Furthermore, the system presents a horizontal symetric behaviour, hence it is likely that we would only need to follow the behaviour of a vortex in the top or bottom and we could predict the behaviour of its counterpart.
```

Sparse sensor placement

Measuring the velocity in the whole domain is technically impossible outside of a simulation. In any experiment, information is gathered from limited sensor measurements. Having gained some insight about the coherent structures existing in the flow, let us leverage these to guide us in placing actual sensors :

1. Choose a desired rank `r` for the PCA truncation (i.e. `Ψ = U[:, 1:r]`)
2. Compute the QR decomposition with pivot of `transpose(Ψ)` using `qr(transpose(Ψ), Val(true))` and return the `r` first pivots `p`.
3. Use the command `plot_flow_field(mesh, U[:, 1], p)` to superimpose the leading PCA mode and the location of the selected sensors.
4. Using your physical intuition, do these sensor locations make sense ? If so, why ? If not, why ?

Rank `r` of the PCA basis : 100

```
md"Rank `r` of the PCA basis : $@bind r Slider(1:100, default=2, show_value=true)"
```

```
Ψ = 20769×100 Array{Float64,2}:
 -3.46018e-6   1.37692e-6  -4.09182e-8   2.16315e-8  …   6.92496e-5    2.68008e-5
 -3.47626e-6   1.3893e-6   -4.03592e-8   2.06432e-8      2.25224e-5    3.04538e-5
 -3.49132e-6   1.40255e-6  -3.97073e-8   1.97293e-8      5.44934e-6    1.87881e-5
 -3.50761e-6   1.4138e-6   -3.9224e-8    1.87091e-8      4.37491e-5    4.99812e-5
 -3.52456e-6   1.424e-6    -3.88307e-8   1.7631e-8       0.00010568    0.000104202
 -3.53885e-6   1.43719e-6  -3.81906e-8   1.67658e-8  …   0.000121789   9.53808e-5
 -3.55466e-6   1.44816e-6  -3.77237e-8   1.57648e-8      5.48673e-5    4.8082e-5
  ⋮                                                  ⋱
  1.56522e-6  -3.0167e-6   -1.78447e-7  -1.4641e-7      -2.61594e-5   -1.50173e-5
  1.55773e-6  -3.00776e-6  -1.58547e-7  -1.25877e-7     -4.43167e-5   -8.65964e-6
  1.54782e-6  -2.99865e-6  -1.4002e-7   -1.12861e-7  …  -6.37844e-5    3.82088e-5
  1.53776e-6  -2.98957e-6  -1.21434e-7  -1.00357e-7     -8.35703e-5    8.69876e-5
  1.52996e-6  -2.97817e-6  -1.09257e-7  -8.27941e-8     -0.000102134   0.000130714
  1.52208e-6  -2.96498e-6  -9.69583e-8  -7.47447e-8     -0.00011932    0.000148214
```

```
# --> Truncated PCA basis Ψ (\Psi <TAB> to have Ψ displayed as variable name).
Ψ= U[:,1:r]
```

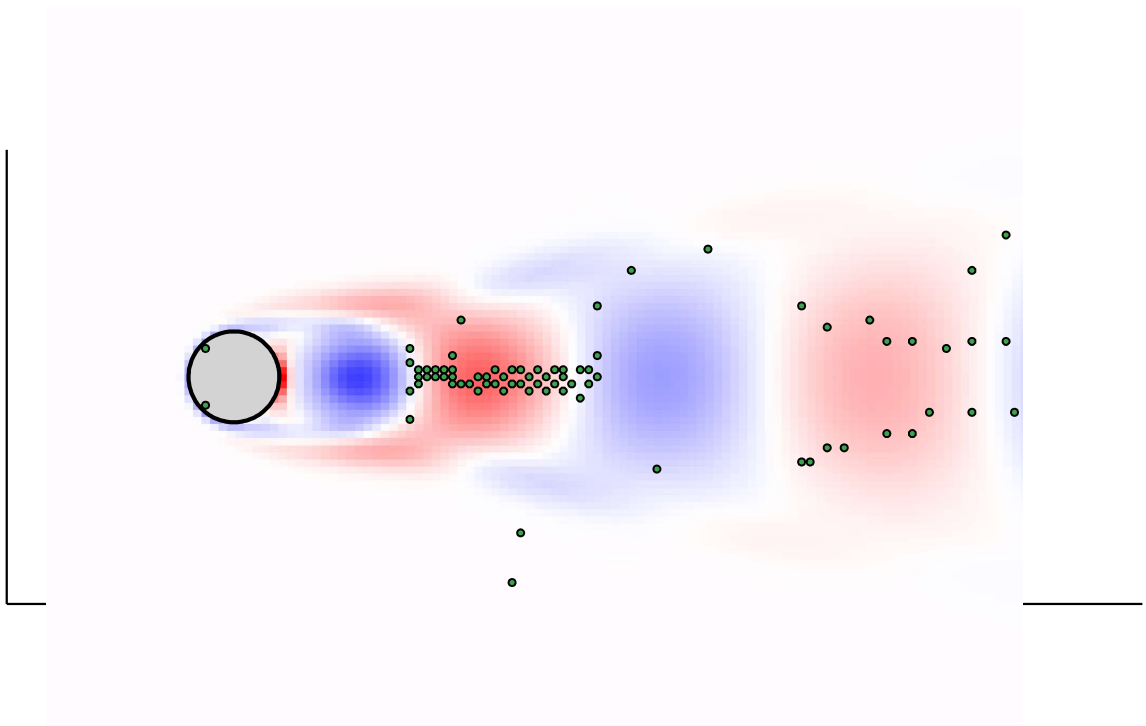
```
 0.0       0.0       0.0          0.0             3.26333e-5    3.16351e-5
 0.0       0.0       0.0          0.0         …  -1.23649e-5   -2.14353e-5
 0.0       0.0       0.0          0.0            -8.29425e-5   -7.72966e-5
 ⋮                                             ⋱
 0.0       0.0       0.0          0.0             6.17759e-5    5.67871e-5
 0.0       0.0       0.0          0.0         …  -3.17262e-5   -2.76707e-5
 0.0       0.0       0.0          0.0            -8.02355e-6   -4.42521e-5
 0.0       0.0       0.0          0.0             0.000227128   0.000144885
 0.0       0.0       0.0          0.0             1.54511e-5    2.84134e-5
 0.0       0.0       0.0          0.0            -0.00021175   -0.000247521
permutation:
```

```
20769-element Array{Int64,1}:
 10597
 11770
 10256
 11761
 10470
 10992
 11130
     ⋮
 20764
 20765
 20766
 20767
 20768
 20769
```

```
# --> Compute the QR decomposition with pivot.
Q , R, ptemp = qr(Ψ',Val(true))
```

```
p =
  Int64[10597, 11770, 10256, 11761, 10470, 10992, 11130, 10772, 10257, 11933, 1008
```

```
p = ptemp[1:r]
```



```
# --> Plot the first PCA mode superimposed with the sensor locations.
plot_flow_field(mesh, U[:, 1], p)
```

```
#Interpretation of sensor locations.
#As mentioned before, the system shows a periodic behaviour which repeats over time and a horizontal symetric component. Also we can observe that as the vortexes get further away from the cylinder, their behaviour is still the same but with a different "Amplitude/Magnitude". Taking this into account, it would make sense that the majority of the sensors are placed in the region where the first Vortex forms, with most of them along the center line, where interactions between the top and bottom vortexes occur, and with a few extra sensors at either the top or bottom halve, just to accurately reconstruct the behaviour of one vortex and predict that of its counter part.
```

Denoting by `a` the projection `tranpose(Ψ) * X` (i.e. the projection of the data into the leading PCA subspace), `y = X[p, :]` the measurements and `Θ = Ψ[p, :]` the measurement matrix, use the least-squares technique to solve

$$\mathbf{y} = \mathbf{\Theta \hat{a}}$$

and compare the the time-series of the estimated `â` coefficients with the true ones in `a` for varying values of `r`. What do you observe ?

```
100×400 Array{Float64,2}:
 -17.8876      -12.2365      -6.41565     -0.49986     …  -44.9831      -44.4366
  38.5281       40.4957      41.7586      42.2799           1.72867       7.48945
  13.0925       13.107       12.1319      10.1578          -6.58142      -3.87504
  -0.641533      1.7782       4.29254      6.77078        -10.7446      -11.0104
  14.1443       14.1965      11.9624       7.83839        -13.8357      -14.8696
  -4.11077       2.15048      7.9414      12.371       …    3.21493      -2.78598
  -2.58558      -0.0794166    2.35091      4.0291           4.68157       4.95491
   ⋮                                                    ⋱
  -1.05073e-7    1.60574e-7  -1.13212e-6   1.83594e-7      -3.61301e-6    7.55756e-8
  -3.63124e-7    9.9103e-7   -2.3478e-6    4.40897e-6   …   3.54117e-6   -9.05945e-7
  -2.12475e-7    6.75981e-7  -2.4823e-6    3.44309e-6      -5.87426e-7    2.05674e-6
   4.70184e-7   -1.27179e-6   3.52149e-6  -8.38028e-6      -4.26927e-6   -4.57986e-7
   2.72952e-7   -6.37113e-7   1.45596e-6  -2.65721e-6      -8.25296e-7    7.6403e-7
  -2.32274e-7    1.03896e-6  -2.29111e-6   4.99833e-6      -3.52149e-6   -4.18717e-6
```

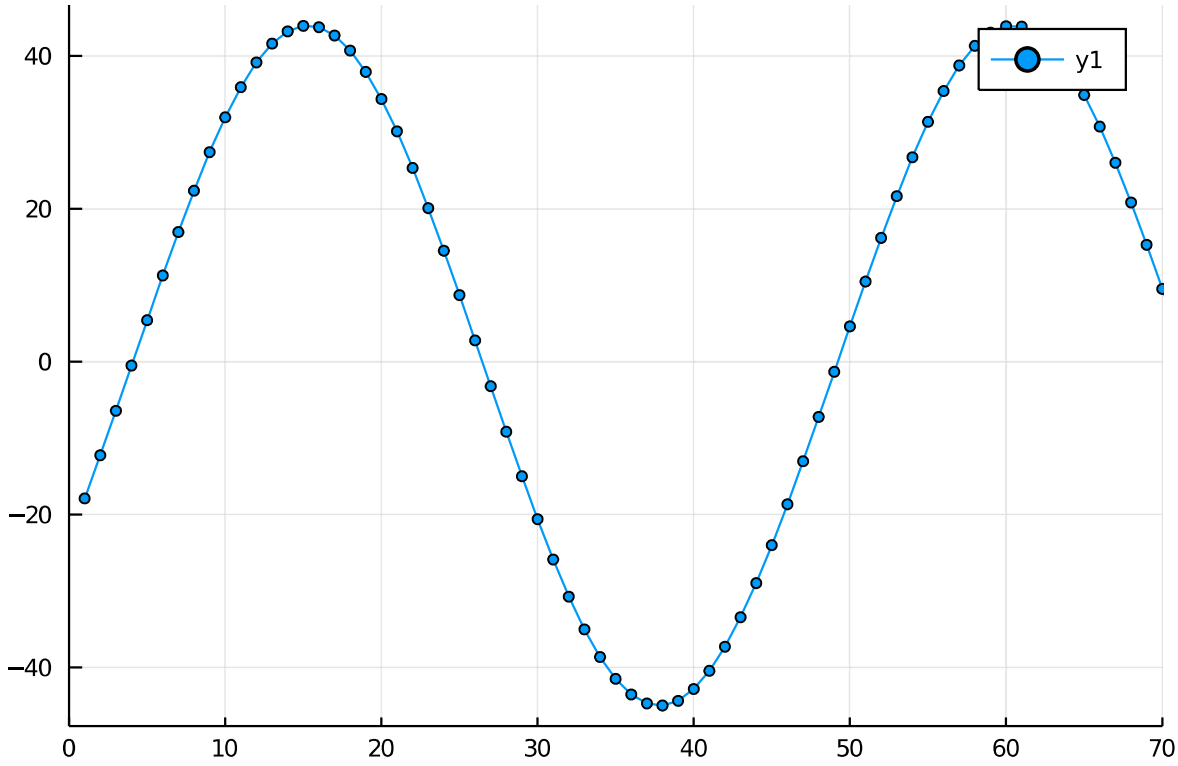
```
begin
    # --> Takes the measurements y.
    y = X[p,:]
    # --> Build the measurement matrix Θ (\Theta <TAB>)
    Θ = Ψ[p,:]

    # --> Obtain the true low-dimensional projection (a = transpose(Ψ) * X).
    a = Ψ' * X

end
```

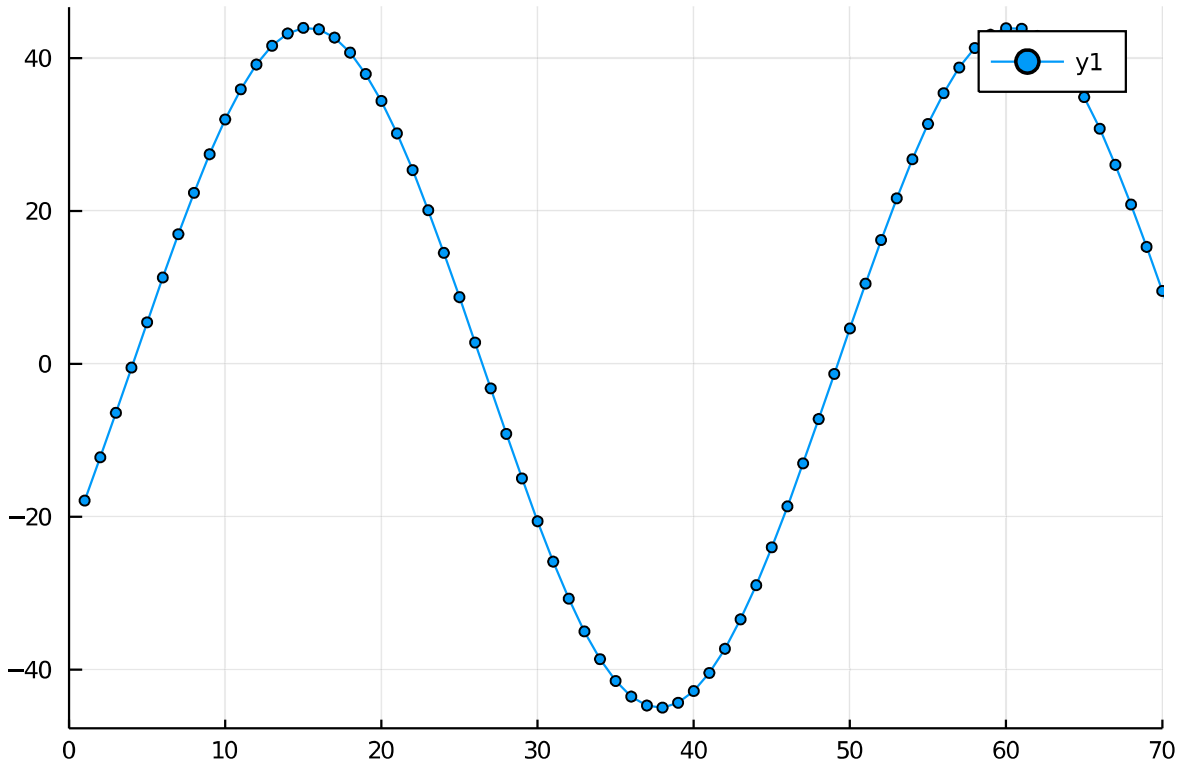
```
ahat =
100×400 Array{Float64,2}:
 -17.8876      -12.2365      -6.41565     -0.499853    …  -44.9831      -44.4366
  38.5281       40.4957      41.7586      42.2799           1.72867       7.48945
  13.0925       13.107       12.1319      10.1578          -6.58143      -3.87505
  -0.641533      1.7782       4.29254      6.77078        -10.7446      -11.0104
  14.1443       14.1965      11.9624       7.83838        -13.8357      -14.8696
  -4.11077       2.15048      7.9414      12.371       …    3.21493      -2.78598
  -2.58558      -0.0794175    2.35091      4.0291           4.68157       4.95491
   ⋮                                                    ⋱
  -3.22387e-7    3.84808e-7  -1.79799e-6   1.33407e-6      -5.5557e-6     2.4655e-6
  -4.79959e-7    1.26931e-6  -3.26117e-6   6.23647e-6   …   3.40404e-6   -6.6055e-7
  -5.58036e-7    1.82074e-6  -4.50034e-6   7.49851e-6      -1.72834e-6    8.98246e-6
   2.94535e-7   -3.4182e-7    1.43791e-6  -3.92655e-6      -8.13642e-6   -2.75845e-6
   4.53325e-7   -1.38541e-6   2.81776e-6  -5.61197e-6       1.00102e-6   -3.91356e-6
  -1.53628e-8    3.19124e-7  -2.65098e-7   9.61107e-7       2.44832e-7   -6.45653e-6
```

```
# --> Solve Θ * â = y.
ahat = Θ\y
```




```
begin
    # --> Plot the time series.
    xaxis = 1:n
    â = transpose(ahat)
    plot(xaxis, â[:,1], marker= (:cicle,3),xlims=(0,70))
    #plot(â[:,1], marker = (:cirle,3), xlims=(0,70))

end
```




```
begin
b = a'
plot(xaxis, b[:,1], marker= (:cicle,3),xlims=(0,70))
    #plot(b[:,1], marker = (:circle,3), xlims=(0,70))
end
```

What do you observe as `r` increases ?`

```
md"
**What do you observe as `r` increases ?`**
" #As we increase the value of r the model is able to represent more accurately the behaviour of our system since there is more information available.
```

Inferring the vorticity field from sparse measurements

Now that we know where to place our sensors and can reconstruct the low-dimensional state vector $\mathbf{a}$, let us recontruct the whole vorticity field. Denoting by $\hat{\mathbf{a}}$ the estimated low-dimensional state vector, the whole flow field can be reconstructed as

$$\hat{\omega}(\mathbf{x}, t_k) = \mathbf{\Psi}\hat{\mathbf{a}}.$$

The reconstruction error is defined as

$$\text{Err} = \frac{1}{N}\sum_{i=1}^{N} \|\hat{\omega}_i - \omega_i\|_2^2$$

Write a function that computes the reconstruction error as a function of the number r of sensors place in the flow.

```
2.5064077647335736e-7
```

```
begin
    w_hat = Ψ * ahat
    w = Ψ * a
    N2 = 0
    for j=1:n
        N2= N2 + (norm(w_hat[:,j]-w[:,j]))^2 #Sum of the squared distances.
    end
    Err = (N2/n)*100 #Computes the error.
end
```

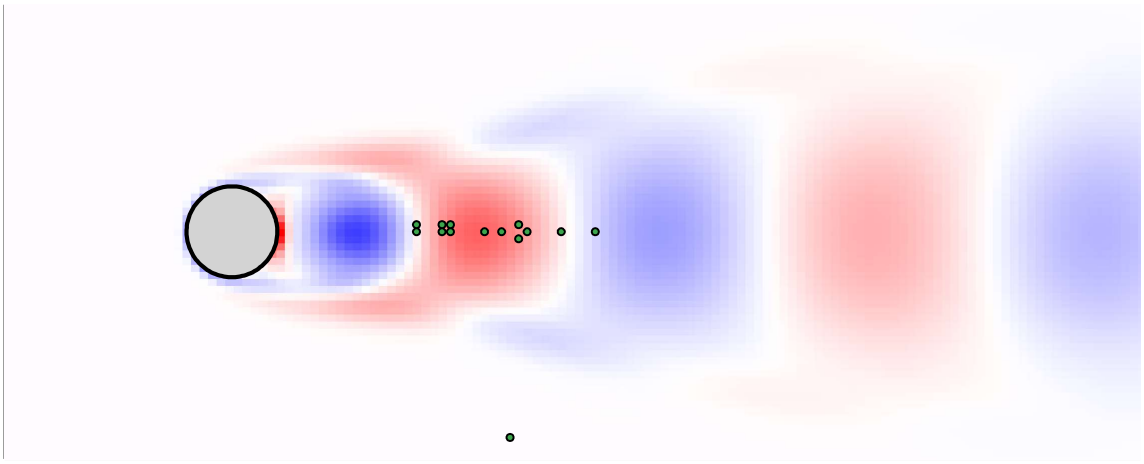
```
21
```

```
#Example of balancing function.
begin
    E = 100
    jj = 1
    r_real = 0 #Starts the count of the number of sensors required.
    tol = 1 #Desired reconstruction accuracy
    while E >= tol
        f_hat = Ψ[:,1:jj] * ahat[1:jj,:]
        f = Ψ[:,1:jj] * a[1:jj,:]
        N3 = 0
            for j = 1:n
                N3 = N3 + (norm(f_hat[:,j]-w[:,j]))^2
            end
        jj = jj + 1
        r_real = r_real + 1 #Adds 1 to the  count at every loop of the while cycle.
        E = (N3/n)*100  #Computes the error at every loop.
    end
    r_real
```

```
end
```

```
Int64[10597, 11770, 10256, 11761, 10470, 10992, 11130, 10772, 10257, 11933, 1008
```

```
p[1:r_real]
```



```
# --> Plot the first PCA mode superimposed with the optimal number of sensors.
plot_flow_field(mesh, U[:, 1], p[1:r_real])
```

Based on your results, how would choose r such that it best balances the reconstruction accuracy and the number of required sensors ? Justify.

```
md"
Based on your results, how would choose $r$ such that it best balances the
reconstruction accuracy and the number of required sensors ?
Justify.
"
```

Answer :

```
md"
**Answer :**
"#I would use a function like the one shown above (Example of balancing function).
Where we would set a limit at the desired accuracy and calculate exactly how many
sensors we would need to get that level of precision.
```

The methodology presented here is quite general and can be applicable to large variety of different engineering fields. Suggest three applications of this methodology to other problems and explain why you think if would be a good way to tackle the problem.

Answer :

```
md"
**Answer :**
```

- "#As seen in class there are multiple applications for this methodology such as Sensor positioning for Climate Prediction/Forcasting, Face Recognition/Reconstruction, and in Quality control of mechanical properties in a machine.
- #In Climate applications it can be used to define the number and location of sensors necessary to create a reliable model used to predict changes in temperature, humidity, wind, etc.
- #For face recognition software it allows us to define the areas of the face where the computer program should focus its attention in order to be able to recognize a face efficiently and also be able to differentiate it form others.
- #For Quality control based on previous experiments performed in a model of a specific machine, we could use this method to place our sensors in the areas where the biggest stresses or deformations tend to be be presented, and use it as a quality check to make sure the machine is able to perform as desired.

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

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

```
plot_flow_field (generic function with 2 methods)
```