Un modello ridotto è un modello che permette di approssimare un problema parametrizzato molto costoso con un modello appunto più piccolo e più semplice da calcolare

Introduciamo il **manifold delle soluzioni** sostanzialmente è l'insieme delle soluzioni a variare del parametro

L'obbiettivo del metodo a base ridotto (RBM) è quello di esploarare il manifold delle soluzioni in diversi punti per trovare un sottospazio di dimensione N molto più piccola rispetto a quella del problema completo, che approssimi bene tutte le soluzioni
Questo sottospazio è formato da **N funzioni di base**, dette **base ridotta**

Questo metodo è diviso in due fasi:
Fase offline (generalmente costosa, ma da eseguire una sola volta per creare il modello ridotto):

- Si esplora il manifold risolvendo alcuni problemi completi
- Da queste soluzioni si costruisce la base ridotta di dimensione N
  Fase online (gneralmente poco costosa):
- Si proietta il nuovo problema per un parametro dato μ, con una procedura detta **di Galerkin**
- Questo consente di avere approssimazioni veloci

Cosa è una procedura di Galerkin?È una procedura che perfette di trasformare un problema continuo in un **sistema lineare**, cioè un problema discreto
Si basa sul **proiettare il problema sull'approssimazione** costruita con le funzioni di base

Senza entrare nel tecnicismo matematico vediamo come si usa dal punto di vista pratico

Prendiamo un problema parametrico:

$$-\frac{d^2u}{dx^2} = \mu$$

In $(0,1)$, $u(0) = u(1) = 0$ dove il parametro è $\mu \in [1,5]$

```
import numpy as np

def truth_problem(N):
    h = 1 / (N + 1)
    A = (2*np.eye(N) - np.eye(N, k=1) - np.eye(N, k=-1)) / h**2
    return A

def termine_noto(mu, N):
```

```
    h = 1 / (N+1)
    return mu * np.ones(N)
```

Spiegazione:
`truth_problem(N)` questa funzione permette di creare la matrice A del sistema lineare $Au = b$ che ci serve per la discretizzazione dell'equazione $-u''(x) = f(x)$
N è il numero di punti su cui discretiziamo il dominio (essendo il dominio $[0, 1]$ avremmo N punti tra 0 e 1)
`h = 1 / (N + 1)` calcoliamo la distanza tra ogni punto scelto
`A = (2*np.eye(N) - np.eye(N, k=1) - np.eye(N, k=-1)) / h**2` questa riga crea la matrice del Laplaciano 1D con differenze finite

`termine_noto(mu, N)` crea il vettore b del sistema $Au = b$ che rappresenta appunto il termine noto

```
def build_reduced_basis(A, mus, N):
    res = []
    for mu in mus:
        b = rhs(mu, N)
        u = np.linalg.solve(A, b)
        res.append(u)
    res_matrix = np.array(res).T

    Q, _ = np.linalg.qr(res_matrix)
    return Q
```

Spiegazione:
`build_reduced_basis` questa funzione serve a craere la base ridotta esplorando il manifold in modo esatto con diversi parametri grazie al metodo delle differenze finite centrali (trova dunque diversi risultati esatti per diversi valori di μ)
Man mano che troviamo soluzioni esatte le mettiamo in un vettore con cui poi creiamo una matrice e calcoliamo la fattorizzazione QR per avere una base ortonormale per maggiore stabilità dopo per la proiezione di Galerkin

```
def solve_online(mu, A, V, N):
    b = rhs(mu, N)
    A_N = V.T @ A @ V
    b_N = V.T @ b
    u_N_coeffs = np.linalg.solve(A_N, b_N)
    u_approx = V @ u_N_coeffs
    return u_approx
```

Spiegazione:
Usiamo il metodo effettivamente per ottenere un'approssimazione partendo dalla base ridotta V , creando e risolvendo il sistema con $A_N = V^T AV$ e $b_n = V^T b$