LUND UNIVERSITY | LTH FACULTY OF ENGINEERING

CFD Uncertainty Quantification

"Intrusive Polynomial Chaos for Uncertainty Quantification of Supersensitivity in the Burgers' Equation Using OpenFOAM"

Javier I. Camacho

Lund University, Sweden

January 21, 2025

Agenda

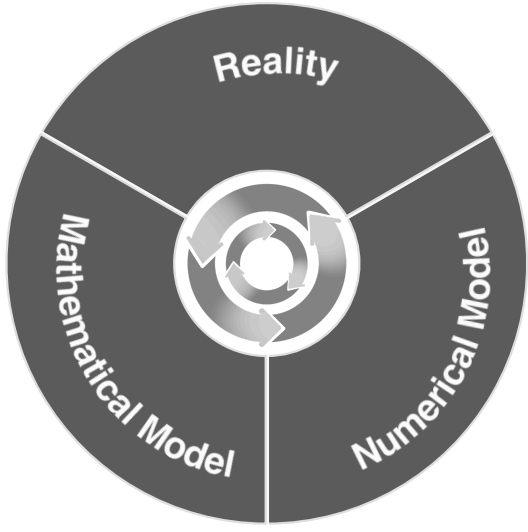
1. **Introduction**
2. **Background**
3. **Existing & Modified Solvers**
4. **Verification & Uncertainty Quantification**
5. **Conclusions & Future Work**

Introduction

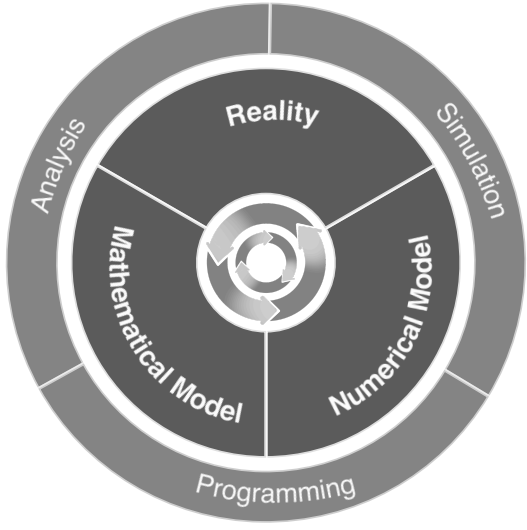
"Essentially, all models are wrong, but some are useful."

— George E.P. Box [1].

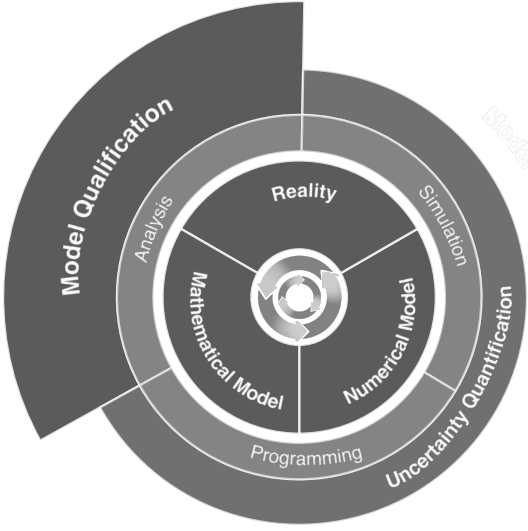
How can we **quantify** how well a **model represents "reality"**?



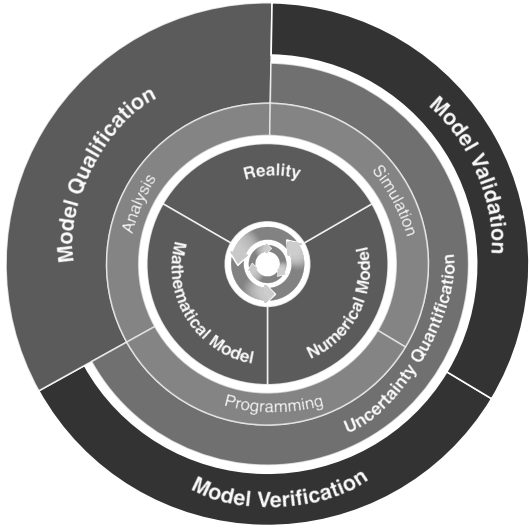
(Diagram inspired from Ref. [2, 3]



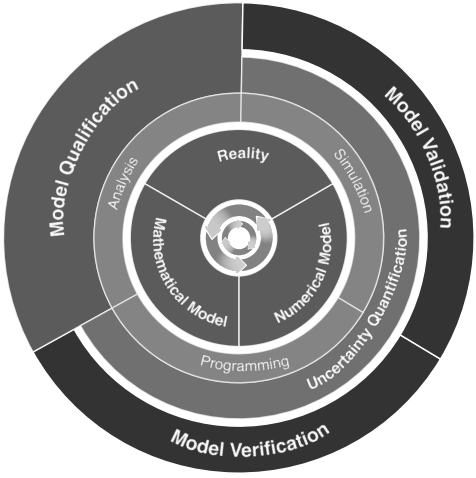
(Diagram inspired from Ref. [2, 3]



(Diagram inspired from Ref. [2, 3]



(Diagram inspired from Ref. [2, 3]

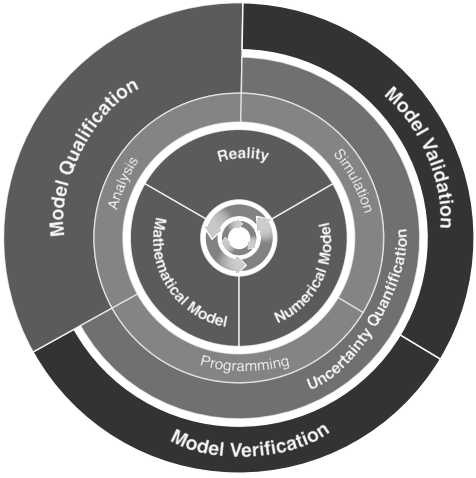


(Diagram inspired from Ref. [2, 3])

Verification

"Are we solving the equations, right?"

- "Verification" $\sim$ solving the equations right.
- "Verification" $\sim$ mathematics



(Diagram inspired from Ref. [2, 3])

Verification

"Are we solving the equations, right?"

- "Verification" $\sim$ solving the equations right.
- "Verification" $\sim$ mathematics

Validation

"Are we solving the right equations?"

- "Validation" $\sim$ solving the right equations.
- "Validation" $\sim$ science/engineering

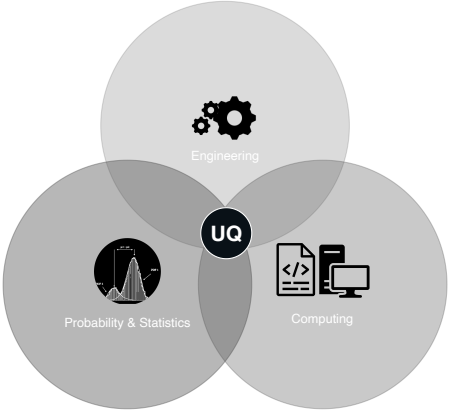
See ASME standards in Ref. [4, 5] for complete details on V&V, alongside Roache's perspective in Ref. [6].

Some Remarks on V&V

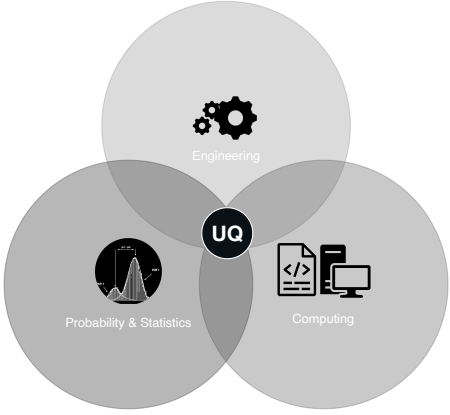
$$\text{Verification:} \begin{cases} \text{Verification of Codes} \\ \text{Verification of Calculations (Solutions)} \end{cases}$$

$$\text{No experimental data} \Rightarrow \text{No validation}^{1}$$

[1] see Ref. [6]



(Diagram inspired from Ref. [7])



(Diagram inspired from Ref. [7])

UQ Key Steps

1. Identifying quantities of interest (QoIs).
2. Modelling uncertainties in inputs.
3. Narrowing down uncertain inputs.
4. Propagating uncertainties through simulations
5. Assessing their impact on the QoIs.

UQ involves several key steps as described by McClarren in Ref. [8].

Intrusive Methods

Methods Examples

- Generalised Polynomial Chaos.
- Perturbation Methods.

Non-Intrusive Methods

Methods Examples

- Monte Carlo Method.
- Non-Intrusive Polynomial Chaos.

Intrusive Methods

Methods Examples

- Generalised Polynomial Chaos.
- Perturbation Methods.

Non-Intrusive Methods

Methods Examples

- Monte Carlo Method.
- Non-Intrusive Polynomial Chaos.

Comparison of Some Relevant Features

Feature	Intrusive Methods	Non-Intrusive Methods
Solver Modification	Required	Not required ("black box")
Flexibility	Limited (solver-specific)	High (any CFD solver)
Computational Cost	Lower for low dimensions	Higher, for the same accuracy of gPC
Ease of Implementation	Complex	Relatively simple

A Word of Caution: Challenges in UQ Theory[2]

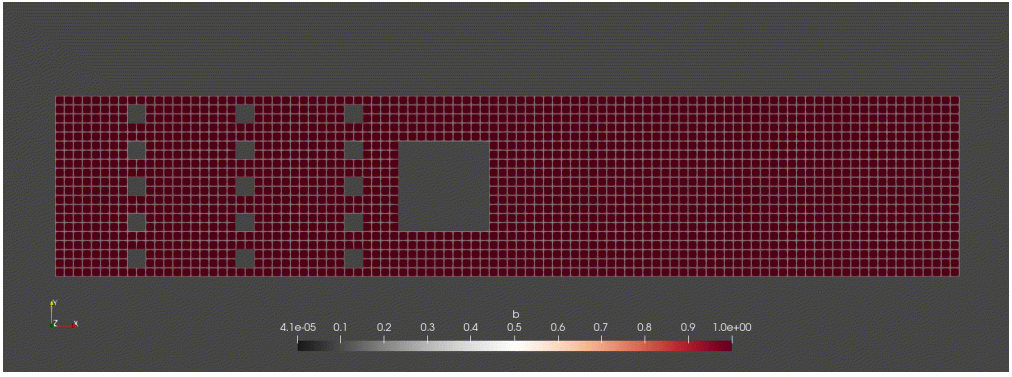
- **UQ is a relatively young discipline**.
- **UQ focuses on solving practical problems** with tailored methods instead of adhering to a unified theoretical framework.
- **A unifying theoretical structure for UQ remains absent**, despite the use of sophisticated methods.

These aspects are still highly relevant today.

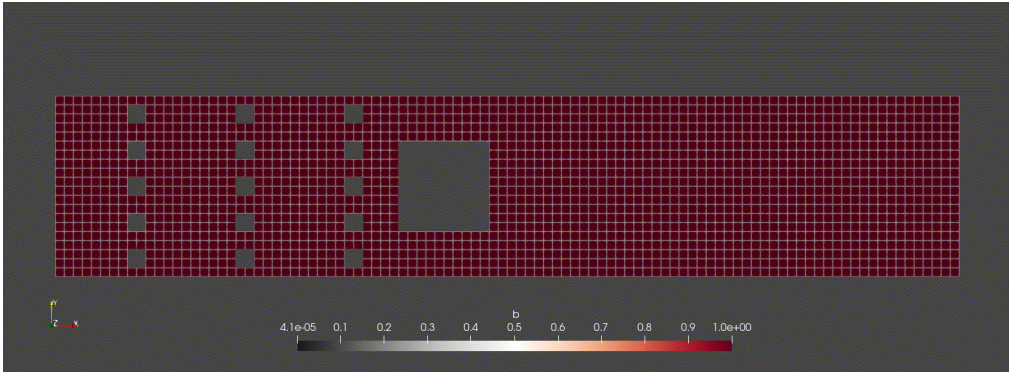
[2] As Sullivan (2015) noted in Ref. [9].

Definition (Fire safety – vocabulary (ISO 13943:2017)[10])

Deflagration: combustion wave propagating at subsonic velocity.

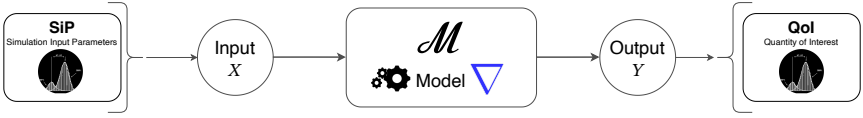


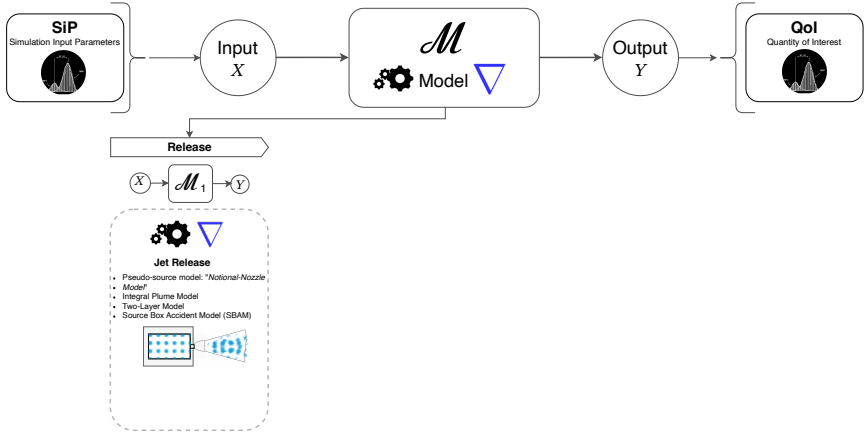
2D-AMR (Adaptive Mesh Refinement) Hydrogen Deflagration Simulation.

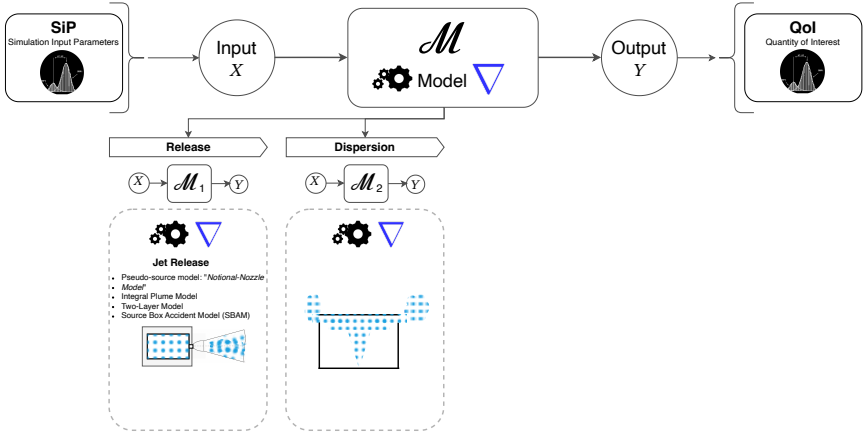


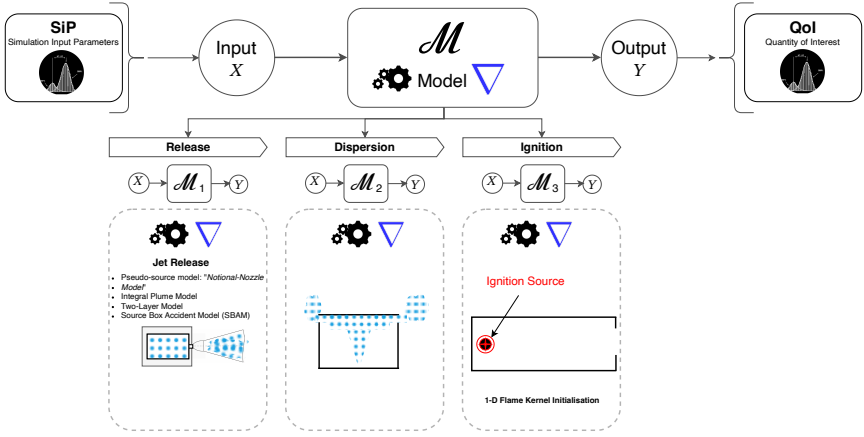
2D-AMR (Adaptive Mesh Refinement) Hydrogen Deflagration Simulation.

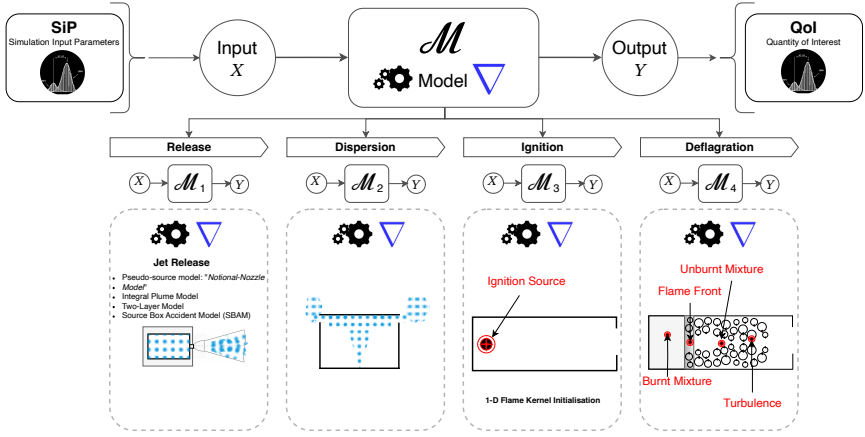
Flame front propagation example, represented by the regression variable $b(0,1)$ (0 : burned, 1 : unburned mixture), in a hydrogen turbulent premixed flame (homogenous flammable mixture with $\phi = 0.7$ (equivalent ratio)) as it propagates through obstacles. The simulation was conducted using a customised `XiFoam` solver within OpenFOAM (v2306) based on the experimental work conducted by Masri et al. [11]. **Formal Verification & Validation (V&V) of the solver is still required.**

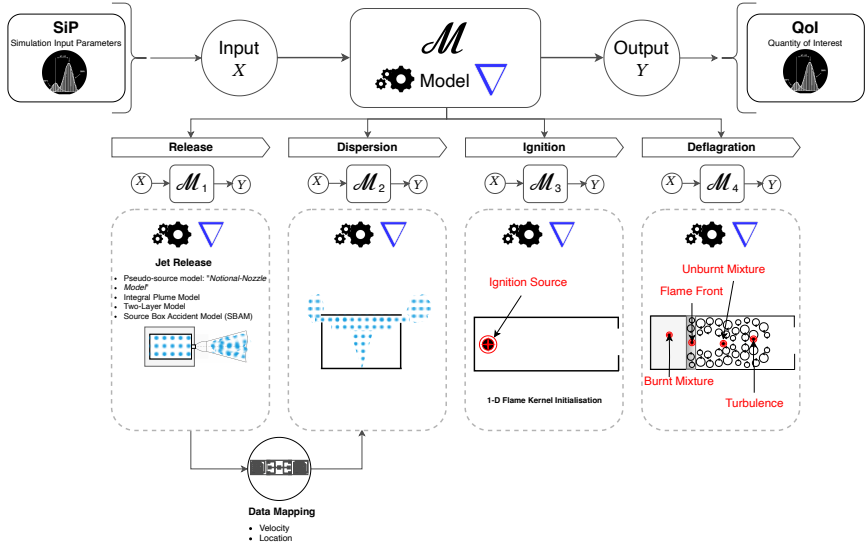

SiP
Simulation Input Parameters
Input
X
$\mathscr{M}$
Model
Output
Y
QoI
Quantity of Interest

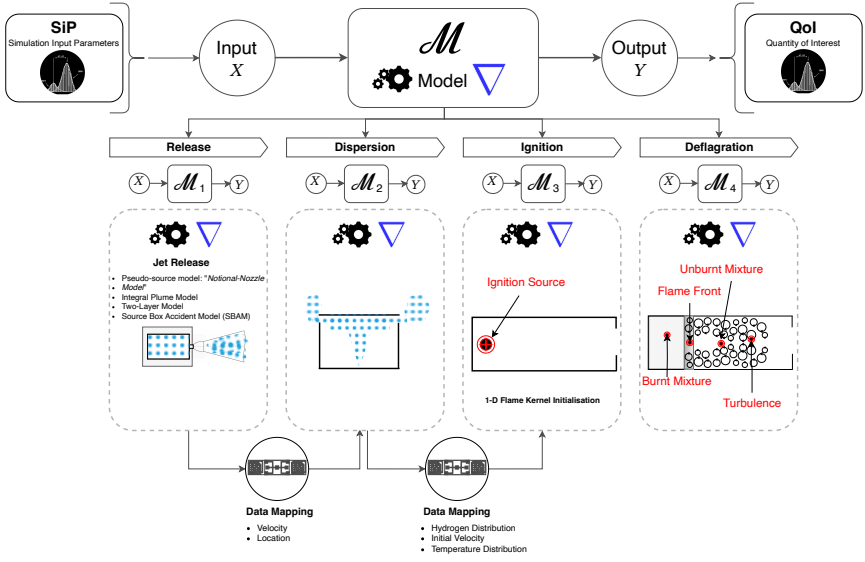

SiP
Simulation Input Parameters
Input
X
$\mathscr{M}$
Model
Output
Y
QoI
Quantity of Interest
Release
X
$\mathscr{M}_1$
Y
Jet Release
Pseudo-source model: "*Notional-Nozzle Model*"
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)

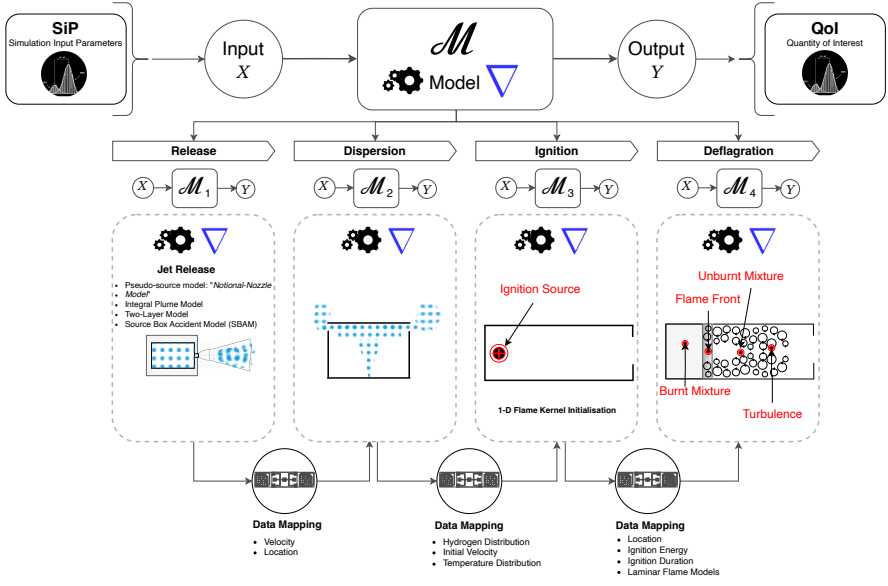

SiP
Simulation Input Parameters
Input
X
$\mathcal{M}$
Model
Output
Y
QoI
Quantity of Interest
Release
Dispersion
X
$\mathcal{M}_1$
Y
X
$\mathcal{M}_2$
Y
Jet Release
Pseudo-source model: "Notional-Nozzle Model"
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)


SiP
Simulation Input Parameters
Input
X
Model
Output
Y
QoI
Quantity of Interest
Release
Dispersion
Ignition
Jet Release
Pseudo-source model: "Notional-Nozzle Model"
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)
Ignition Source
1-D Flame Kernel Initialisation


SiP
Simulation Input Parameters
Input
X
Model
Output
Y
QoI
Quantity of Interest
Release
Dispersion
Ignition
Deflagration
Jet Release
Pseudo-source model: 'Notional-Nozzle Model'
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)
Ignition Source
1-D Flame Kernel Initialisation
Unburnt Mixture
Flame Front
Burnt Mixture
Turbulence




SiP
Simulation Input Parameters
Input
X
Model
Output
Y
QoI
Quantity of Interest
Release
Dispersion
Ignition
Deflagration
Jet Release
Pseudo-source model: "Notional-Nozzle Model"
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)
Ignition Source
1-D Flame Kernel Initialisation
Unburnt Mixture
Flame Front
Burnt Mixture
Turbulence
Data Mapping
Velocity
Location
Data Mapping
Hydrogen Distribution
Initial Velocity
Temperature Distribution


SiP
Simulation Input Parameters
Input
X
Model
Output
Y
QoI
Quantity of Interest
Release
Dispersion
Ignition
Deflagration
Jet Release
Pseudo-source model: "Notional-Nozzle Model"
Integral Plume Model
Two-Layer Model
Source Box Accident Model (SBAM)
Ignition Source
1-D Flame Kernel Initialisation
Unburnt Mixture
Flame Front
Burnt Mixture
Turbulence
Data Mapping
Velocity
Location
Data Mapping
Hydrogen Distribution
Initial Velocity
Temperature Distribution
Data Mapping
Location
Ignition Energy
Ignition Duration
Laminar Flame Models

Objectives

- Provide Background on *Generalised Polynomial Chaos Method.*
- Tutorial Development on Solver Modifications & Verification.
- Improving Existing Solvers for Adaptability and Scalability.

Scope

- Specific Problem on Viscous Burgers' equation *"Supersensitivity".*[3]
- Existing Solvers `burgersFoam` and `gPCBurgersFoam` Modifications.
- Pre- and Post-Processing Tools.
- Verification & Uncertainty Quantification.

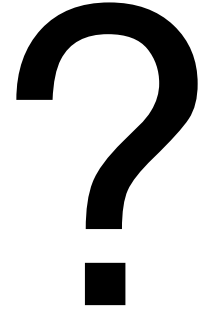
[3] Building upon Xiu and Karniadakis's work on "Supersensitivity due to Uncertain Boundary Conditions" (Ref. [12]).

Background

What is the problem

Why this problem

How we approach it



1D Viscous Burgers Equation (non-conservative form Eq. (1))

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in \mathbb{R}, \quad t \geq 0. \tag{1}$$

[4] For further details see Ref. [13]

1D Viscous Burgers Equation (non-conservative form Eq. (1))

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in \mathbb{R}, \quad t \geq 0. \tag{1}$$

Modelling Versatility of Burgers' Equation[4]

- Well known exact solution of the partial differential equation.
- Modelling nonlinear acoustic waves and shock.
- Used as standard test problem for the PDE solvers evaluation.
- Diverse applications fields (e.g., Turbulence modelling, shock wave theory, Traffic flow and gas dynamics).

[4] For further details see Ref. [13]

1D Viscous Burgers Equation (conservative form Eq. (2))

$$\frac{\partial u}{\partial t} + \frac{\partial}{\partial x}\left(\frac{1}{2}u^2\right) = \nu \frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1], \quad t \geq 0. \tag{2}$$

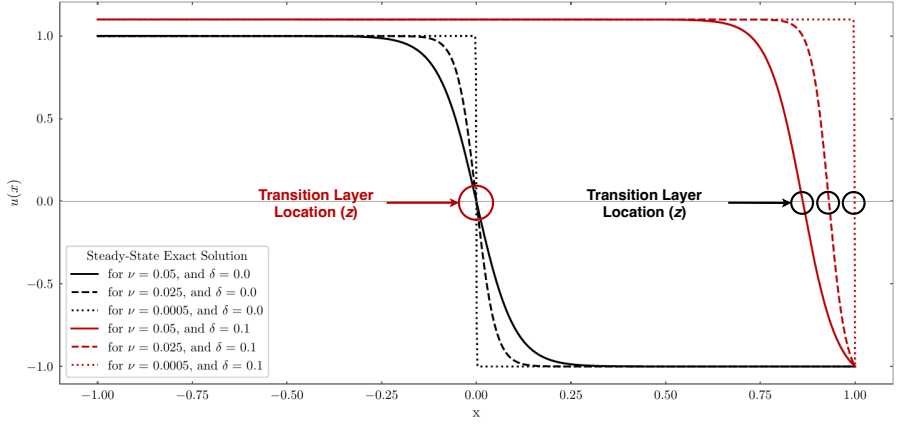
Boundary and Initial Conditions (Eq. 3, Eq. 4)

$$u(-1, t) = 1 + \delta, \quad u(1, t) = -1, \quad t \geq 0, \tag{3}$$

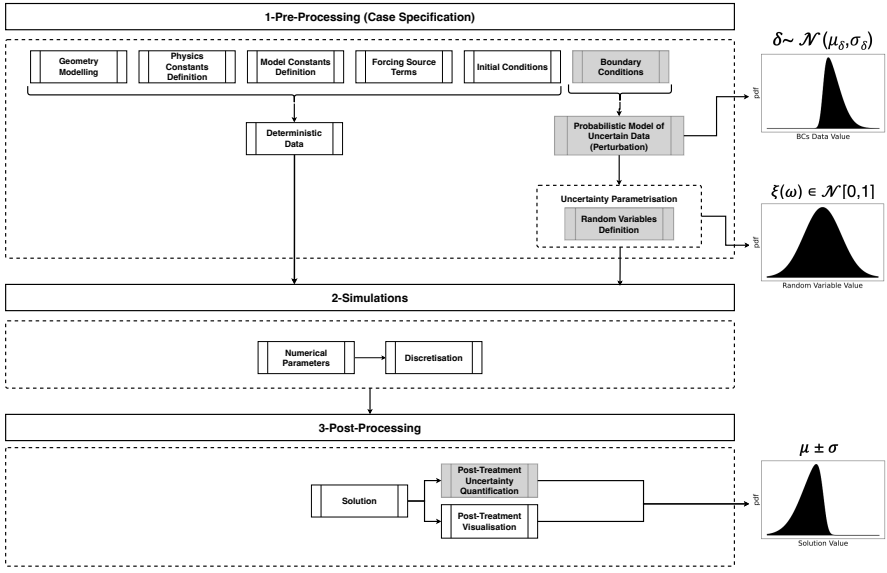
$$u(x, 0) = 0, \quad x \in [-1, 1], \tag{4}$$

Small imposed perturbation (δ) is assumed to be a $\delta \in (0, \epsilon) \sim \mathcal{U}(0, \epsilon)$ uniform distribution between 0 and ϵ.

Transition Layer Location (z) and Supersensitivity Phenomenon



Further details on steady-state exact solution see report "Intrusive Polynomial Chaos for Uncertainty Quantification of Supersensitivity in the Burgers' Equation Using OpenFOAM" in Ref. [14].



Generalised Polynomial Chaos Expansion

$$u(x,t;\omega) = \sum_{i=0}^{\infty} \hat{u}_i(x,t)\Phi_i(\xi(\omega)), \tag{5}$$

Generalised Polynomial Chaos Expansion

$$u(x,t;\omega) = \sum_{i=0}^{\infty} \hat{u}_i(x,t)\Phi_i(\xi(\omega)), \tag{5}$$

Polynomial Truncation

$$u(x,t;\omega) \approx \tilde{u}(x,t;\omega) = \sum_{i=0}^{P} \hat{u}_i(x,t)\Phi_i(\xi(\omega)), \tag{6}$$

Generalised Polynomial Chaos Expansion

$$u(x,t;\omega) = \sum_{i=0}^{\infty} \hat{u}_i(x,t)\Phi_i(\xi(\omega)), \tag{5}$$

Polynomial Truncation

$$u(x,t;\omega) \approx \tilde{u}(x,t;\omega) = \sum_{i=0}^{P} \hat{u}_i(x,t)\Phi_i(\xi(\omega)), \tag{6}$$

where

$\xi(\omega)$: Multi-dimensional random variables dependent on ω.

$\Phi_i(\xi(\omega))$: Orthogonal polynomial basis functions over the probability space of ξ.

$\hat{u}_i(x,t)$: Deterministic coefficients or modes.

P: Highest polynomial order.

Polynomial Basis Orthogonality Property

$$\langle \Phi_i(\xi), \Phi_j(\xi) \rangle = \int_S \Phi_i(\xi)\Phi_j(\xi)\rho(\xi)d\xi = \gamma_i \delta_{ij}, \text{with} \quad \mathrm{i,j} \in \mathbb{N}, \tag{7}$$

Polynomial Basis Orthogonality Property

$$\langle \Phi_i(\xi), \Phi_j(\xi) \rangle = \int_S \Phi_i(\xi)\Phi_j(\xi)\rho(\xi)d\xi = \gamma_i \delta_{ij}, \text{with} \quad \mathrm{i,j} \in \mathbb{N}, \tag{7}$$

$$\gamma_i = \mathbb{E}[\Phi_i(\xi)^2] = \langle \Phi_i^2 \rangle = \int_S \Phi_i^2(\xi)\rho(\xi)d\xi, \tag{8}$$

where

- $\rho(\xi)$: Weighting function (PDF of the random variable ξ).
- δ_{ij}: Kronecker delta.
- $\Phi_i(\xi)$, $\Phi_j(\xi)$: Orthogonal polynomial basis functions.
- $\langle \cdot, \cdot \rangle$: Inner product, also called ensemble average or expectation ($\mathbb{E}[\cdot]$).
- γ_i: Normalization constant (equals 1 if basis functions are normalized).
- S: Support of $\rho(\xi)$, depending on the polynomial basis.

Expanded Random Burgers' Equation

$$\frac{\partial}{\partial t}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)+\left(\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)\right)\frac{\partial}{\partial x}\left(\sum_{j=0}^{P}\hat{u}_j(x,t)\Phi_j(\xi)\right)=\nu\frac{\partial^2}{\partial x^2}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi). \quad (9)$$

[5] For more details, see Chapter 17 *"Intrusive Polynomial Chaos Methods for Forward Uncertainty Propagation"* in Ref. [15].

Expanded Random Burgers' Equation

$$\frac{\partial}{\partial t}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)+\left(\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)\right)\frac{\partial}{\partial x}\left(\sum_{j=0}^{P}\hat{u}_j(x,t)\Phi_j(\xi)\right)=\nu\frac{\partial^2}{\partial x^2}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi). \quad (9)$$

Galerkin Projection

$$\langle\mathcal{L}[\tilde{u}(x,t,\omega)],\Phi_k(\xi(\omega))\rangle=0,\quad k=0,\ldots,P, \quad (10)$$

[5] For more details, see Chapter 17 *"Intrusive Polynomial Chaos Methods for Forward Uncertainty Propagation"* in Ref. [15].

Expanded Random Burgers' Equation

$$\frac{\partial}{\partial t}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)+\left(\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)\right)\frac{\partial}{\partial x}\left(\sum_{j=0}^{P}\hat{u}_j(x,t)\Phi_j(\xi)\right)=\nu\frac{\partial^2}{\partial x^2}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi). \quad (9)$$

Galerkin Projection

$$\langle\mathcal{L}[\tilde{u}(x,t,\omega)],\Phi_k(\xi(\omega))\rangle=0,\quad k=0,\ldots,P, \quad (10)$$

where $\mathcal{L}[\cdot]$ is the differential operator. Using orthogonality, the final formulation yields coupled deterministic equations for the expansion coefficients $\hat{u}_k(x,t)$.

[5] For more details, see Chapter 17 *"Intrusive Polynomial Chaos Methods for Forward Uncertainty Propagation"* in Ref. [15].

Expanded Random Burgers' Equation

$$\frac{\partial}{\partial t}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)+\left(\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi)\right)\frac{\partial}{\partial x}\left(\sum_{j=0}^{P}\hat{u}_j(x,t)\Phi_j(\xi)\right)=\nu\frac{\partial^2}{\partial x^2}\sum_{i=0}^{P}\hat{u}_i(x,t)\Phi_i(\xi). \quad (9)$$

Galerkin Projection

$$\langle\mathcal{L}[\tilde{u}(x,t,\omega)],\Phi_k(\xi(\omega))\rangle=0,\quad k=0,\ldots,P, \quad (10)$$

where $\mathcal{L}[\cdot]$ is the differential operator. Using orthogonality, the final formulation yields coupled deterministic equations for the expansion coefficients $\hat{u}_k(x,t)$.

Key Property: The residual $\epsilon = u - \tilde{u}$ is orthogonal to the space spanned by the basis functions. This ensures the best approximation in the L_2-norm sense[5].

[5] For more details, see Chapter 17 *"Intrusive Polynomial Chaos Methods for Forward Uncertainty Propagation"* in Ref. [15].

Deterministic System of Coupled Equations

$$\frac{\partial \hat{u}_k}{\partial t} + \frac{1}{2}\sum_{i=0}^{P}\sum_{j=0}^{P}\frac{\partial(\hat{u}_i\hat{u}_j)}{\partial x}M_{ijk} = \nu\frac{\partial^2\hat{u}_k}{\partial x^2}, \quad \forall k \in [0, P], \tag{11}$$

System of P+1 coupled equations

Deterministic System of Coupled Equations

$$\frac{\partial \hat{u}_k}{\partial t} + \frac{1}{2}\sum_{i=0}^{P}\sum_{j=0}^{P}\frac{\partial(\hat{u}_i\hat{u}_j)}{\partial x}M_{ijk} = \nu\frac{\partial^2\hat{u}_k}{\partial x^2}, \quad \forall k \in [0, P], \tag{11}$$

System of P+1 coupled equations

Galerkin Tensor (3rd order) (Pre-Processing Phase)

$$M_{ijk} = \frac{e_{ijk}}{\gamma_k} = \frac{\langle \Phi_i \Phi_j \Phi_k \rangle}{\langle \Phi_k^2 \rangle} \tag{12}$$

Some Properties of M_{ijk}

Sparsity: many M_{ijk} entries are zero due to orthogonality, improving efficiency.

Pre-computation: M_{ijk} can be precomputed and reused, reducing costs.

Numerical Evaluation: Often computed using Gaussian quadrature for practical use.

Left Boundary Conditions Expansion (Pre-Processing Phase)

$$\tilde{u}(-1) = 1 + \boldsymbol{\delta}, \tag{13}$$

$$\tilde{u}(-1) = \sum_{k=0}^{P} \hat{u}_k(-1)\Phi_k(\xi), \quad \forall k \in [0, P]. \tag{14}$$

Small imposed perturbation (δ) is assumed to be a $\delta \in (0, \epsilon) \sim \mathcal{U}(0, \epsilon)$ (uniform distributed).

Left Boundary Conditions Expansion (Pre-Processing Phase)

$$\tilde{u}(-1) = 1 + \boldsymbol{\delta}, \tag{13}$$

$$\tilde{u}(-1) = \sum_{k=0}^{P} \hat{u}_k(-1)\Phi_k(\xi), \quad \forall k \in [0, P]. \tag{14}$$

Small imposed perturbation (δ) is assumed to be a $\delta \in (0, \epsilon) \sim \mathcal{U}(0, \epsilon)$ (uniform distributed).

Legendre Polynomial Expressions

$$\Phi_0(\xi) = 1, \quad \Phi_1(\xi) = \xi, \quad \Phi_2(\xi) = \frac{1}{2}(3\xi^2 - 1), \ldots \tag{15}$$

Apply the First-Order Polynomial Chaos Expansion

$$\tilde{u}(-1) = \hat{u}_0(-1)\Phi_0(\xi) + \hat{u}_1(-1)\Phi_1(\xi), \tag{16}$$

$$= \hat{u}_0(-1) + \hat{u}_1(-1)\xi. \tag{17}$$

Given a uniformly distributed perturbation δ with support $(0, \epsilon)$, the values of μ_δ and σ_δ are derived directly from their definitions [6].

[6] By definition from page 333-334 (A.11 Uniform Distribution A.11.3 Properties) in Ref. [8].

Given a uniformly distributed perturbation δ with support $(0, \epsilon)$, the values of μ_δ and σ_δ are derived directly from their definitions [6].

Compute Expansion Deterministic Coefficients/Modes

$$\mu_\delta = \frac{a+b}{2} = \frac{0+\epsilon}{2} = \frac{\epsilon}{2}, \tag{18}$$

$$\sigma_\delta = \frac{b-a}{2\sqrt{3}} = \frac{\epsilon-0}{2\sqrt{3}} = \frac{\epsilon}{2\sqrt{3}}. \tag{19}$$

[6] By definition from page 333-334 (A.11 Uniform Distribution A.11.3 Properties) in Ref. [8].

Given a uniformly distributed perturbation δ with support $(0, \epsilon)$, the values of μ_δ and σ_δ are derived directly from their definitions [6].

Compute Expansion Deterministic Coefficients/Modes

$$\mu_\delta = \frac{a+b}{2} = \frac{0+\epsilon}{2} = \frac{\epsilon}{2}, \tag{18}$$

$$\sigma_\delta = \frac{b-a}{2\sqrt{3}} = \frac{\epsilon-0}{2\sqrt{3}} = \frac{\epsilon}{2\sqrt{3}}. \tag{19}$$

The expansion coefficients can be expressed as follows

$$\hat{u}_0(-1) = 1 + \mu_\delta = 1 + \frac{\epsilon}{2}, \tag{20}$$

$$\hat{u}_1(-1) = \sigma_\delta = \frac{\epsilon}{2\sqrt{3}}. \tag{21}$$

[6] By definition from page 333-334 (A.11 Uniform Distribution A.11.3 Properties) in Ref. [8].

Left Boundary Conditions

$$\hat{u}_k(-1) = \begin{cases} 1 + \mu_\delta = 1 + \frac{\epsilon}{2}, & \text{if } k = 0, \\ \sigma_\delta = \frac{\epsilon}{2\sqrt{3}}, & \text{if } k = 1, \\ 0, & \text{if } k \geq 2. \end{cases} \tag{22}$$

Right Boundary Condition

$$\hat{u}_k(1) = \begin{cases} -1, & \text{if } k = 0, \\ 0, & \text{if } k \geq 1. \end{cases} \tag{23}$$

Statistical Moments of the Solution (Post-Processing Phase)

Expected or Mean Value (μ)

$$\mathbb{E}[\tilde{u}(x,t,\omega)] = \langle \tilde{u}(x,t,\omega) \rangle = \sum_{k=0}^{P} \hat{u}_k(x,t) \langle \Phi_0, \Phi_k \rangle = \hat{u}_0(x,t). \tag{24}$$

Statistical Moments of the Solution (Post-Processing Phase)

Expected or Mean Value (μ)

$$\mathbb{E}[\tilde{u}(x,t,\omega)] = \langle \tilde{u}(x,t,\omega) \rangle = \sum_{k=0}^{P} \hat{u}_k(x,t) \langle \Phi_0, \Phi_k \rangle = \hat{u}_0(x,t). \tag{24}$$

Variance (σ^2)

$$\mathbb{V}[\tilde{u}(x,t,\omega)] = \sigma^2 = \mathbb{E}\left[(\tilde{u}(x,t,\omega) - \mathbb{E}[\tilde{u}(x,t,\omega)])^2 \right] = \sum_{k=1}^{P} \hat{u}_k^2(x,t) \langle \Phi_k^2 \rangle = \sum_{k=1}^{P} \hat{u}_k^2(x,t). \tag{25}$$

Existing & Modified Solvers

What challenges

do solvers face for UQ

How do we modify

an existing solver

```
while (simple.loop())
{
    Info<< "Time = " << runTime.timeName() << nl <<
  endl;

    while (simple.correctNonOrthogonal())
    {
        forAll(Uhat, k)
        {
            fvVectorMatrix UhatEqn(
                fvm::ddt(Uhat[k])
                -
                fvm::laplacian(nu, Uhat[k])
            );

            forAll(Uhat, i){
                forAll(Uhat, j){
                    if(j == k)
                        UhatEqn += e[i][j][k] * fvm
  ::div(phihat[i], Uhat[j]);
                    else
                        UhatEqn += e[i][j][k] * fvc
  ::div(phihat[i], Uhat[j]);
                }
            }
            solve(UhatEqn);
            phihat[k] = linearInterpolate(Uhat[k]) &
   mesh.Sf();
        }

    }
    runTime.write();
}
```

```
while (simple.loop())
{
    Info<< "Time = " << runTime.timeName() << nl <<
   endl;

    while (simple.correctNonOrthogonal())
    {
        forAll(Uhat, k)
        {
            fvVectorMatrix UhatEqn(
                fvm::ddt(Uhat[k])
                -
                fvm::laplacian(nu, Uhat[k])
            );

            forAll(Uhat, i){
                forAll(Uhat, j){
                    if(j == k)
                        UhatEqn += e[i][j][k] * fvm
  ::div(phihat[i], Uhat[j]);
                    else
                        UhatEqn += e[i][j][k] * fvc
  ::div(phihat[i], Uhat[j]);
                }
            }
            solve(UhatEqn);
            phihat[k] = linearInterpolate(Uhat[k]) &
   mesh.Sf();
        }

    }
    runTime.write();
}
```

Implemented Equation in gPCBurgersFoam.C

$$\frac{\partial \hat{u}_k}{\partial t} + \sum_{i=0}^{P}\sum_{j=0}^{P} \frac{\partial(\hat{u}_i \hat{u}_j)}{\partial x} e_{ijk} = \nu \frac{\partial^2 \hat{u}_k}{\partial x^2}, \quad \forall k \in [0, P].$$

Same as Eq. (11), but $\frac{1}{2}$ factor is missing.

```
Info<< "\nCalculating gPC 1D-Burgers Equation\n" << endl;
while (simple.loop())
{
    Info<< "Time = " << runTime.timeName() << nl << endl;

    while (simple.correctNonOrthogonal())
    {
        forAll(Uhat, k)
        {
            fvVectorMatrix UhatEqn
            (
                fvm::ddt(Uhat[k])
              -
                fvm::laplacian(nu, Uhat[k])
            );

            forAll(Uhat, i)
            {
                forAll(Uhat, j)
                {
                    if(j == k)
                    {
                        UhatEqn += (*e)[i][j][k] * 0.5 *
  fvm::div(phihat[i], Uhat[j]);
                    }
                    else
                    {
                        UhatEqn += (*e)[i][j][k] * 0.5 *
  fvc::div(phihat[i], Uhat[j]);
                    }
                }
            }
            UhatEqn.relax();
            UhatEqn.solve();
            phihat[k] = fvc::flux(Uhat[k]);
```

Main Implemented Modifications in myGPCBurgersFoam.C

- Adding $\frac{1}{2}$ factor, to align with viscous Burgers' equation in its conservative form.
- Smart pointer `(*e)[i][j][k]`.
- Using `phihat[k] = fvc::flux(Uhat[k])` instead of `phihat[k] =linearInterpolate (Uhat[k]) & mesh.Sf()`.
- Add flexibility (`UhatEqn.relax()`) and keep OpenFOAM style (`UhatEqn.solve()`).

```
scalar e[3][3][3];
e[0][0][0] = 1.0;
e[1][1][0] = 1.0;
e[2][2][0] = 1.0;

e[0][1][1] = 1.0;
e[1][0][1] = 1.0;
e[1][2][1] = Foam::sqrt(2.0);
e[2][1][1] = Foam::sqrt(2.0);

e[0][2][2] = 1.0;
e[2][0][2] = 1.0;
e[1][1][2] = Foam::sqrt(2.0);
e[2][2][2] = 2.0 * Foam::sqrt(2.0);
```

Main Drawbacks in the Implemented `createFields.H` (Tensor Coefficients)

- **Hardcoded Values**: Hardcoded polynomial type/order for precomputed tensor coefficients `e[i][j][k]`.
- **Recompilation**: Requires recompilation for changes in random variable distribution or polynomial order.
- **Uninitialized**: zero components in `e[i][j][k]` tensor can crash the solver due to arbitrary values assigned by OpenFOAM.

```
Info<< "Initialize Galerkin Tensor based on order\n" << endl;
autoPtr<Foam::List<Foam::List<Foam::List<scalar>>>> e(
    new Foam::List<Foam::List<Foam::List<scalar>>>(
        order + 1,
        Foam::List<Foam::List<scalar>>(
            order + 1,
            Foam::List<scalar>(
                order + 1, 0.0
            )
        )
    )
);
```

Main Implemented Modifications
createFields.H
(Tensor Coefficients Initialisation)

- **Memory Management with** `autoPtr`: Smart pointer for automatic memory allocation and cleanup, preventing leaks and simplifying memory management.
- **Tensor Structure**: `e[i][j][k]` is dynamically allocated with `Foam::List` and initialized to zero, ensuring flexibility and avoiding arbitrary default values.

```
Info<< "Reading gPCCoeff\n" << endl;
    IOdictionary gPCCoeff
    (
        IOobject
        (
            "gPCCoeff",
            runTime.constant(),
            mesh,
            IOobject::MUST_READ,
            IOobject::NO_WRITE
        )
    );

    for (int i = 0; i <= order; ++i)
    {
        for (int j = 0; j <= order; ++j)
        {
            for (int k = 0; k <= order; ++k)
            {
                word entryName = "e[" + Foam::name(i) + "][" +
                Foam::name(j) + "][" + Foam::name(k) + "]";
                if (gPCCoeff.found(entryName))
                {
                    // Assign value to the tensor
                        (*e)[i][j][k] = readScalar(gPCCoeff.
      lookup(entryName));
                    // Verification: Print the entry name and
      value
                    Info << "Loaded coefficient " << entryName
       << ": "
                        << (*e)[i][j][k] << endl;
```

Main Implemented Modifications `createFields.H` (Reading Precomputed Tensor Coeff.)

- `gPCCoeff`: Reads precomputed Galerkin coefficients from the `gPCCoeff` file, created during pre-processing and saved in the `constant` directory.
- The nested loop structure dynamically reads and populates the Galerkin tensor `e[i][j][k]`. Chosen for clarity and practicality.
- Future optimizations could focus on more compact or performance-oriented solutions.

Pre-Processing Main Steps[7]

Input Data: Read `UQProperties` dictionary.

Step 1: Generate Distribution and Polynomials.

Step 2: Orthonormality Verification.

Step 3: Calculate Tensor Coefficients.

Step 4: Galerkin Coefficient Verification.

Output Data: Write output file (`gPCCoeff`) and save to the `constant` directory.

Post-Processing Tools

For details on Post-Processing Tools see report "Intrusive Polynomial Chaos for Uncertainty Quantification of Supersensitivity in the Burgers' Equation Using OpenFOAM" in Ref. [14].

[7] **Required Tools**: `pyFoam` **install it separately [16], to install use:** `pip install PyFoam`. `chaospy` **installation link[17], to install use:** `pip install chaospy`.

Step 1: Generate Distribution and Polynomials

```
def generate_distribution_and_polynomials(
    order, poly_type):
    if poly_type.lower() == "legendre":
        lower_std = -1 # standard lower bound
        upper_std = 1  # standard upper bound
        distribution = cp.Uniform(lower=
    lower_std, upper=upper_std)
        polynomials = cp.expansion.legendre(
    order, lower=lower_std,

         upper=upper_std,

         physicist=False,

         normed=True)
    else:
        raise ValueError("Unsupported
    polynomial type. Use 'legendre'.")
    return distribution, polynomials
```

Function Overview

- Creates a standardised uniform distribution ($[-1, 1]$) for ξ and generates corresponding **Legendre polynomials** (Φ) based on the specified order.
- Ensures consistency with the orthogonality properties of Legendre polynomials.
- Designed for extensibility to include other polynomial types (e.g., **Hermite**) with conditional logic.

Step 3: Calculate Tensor Coefficients

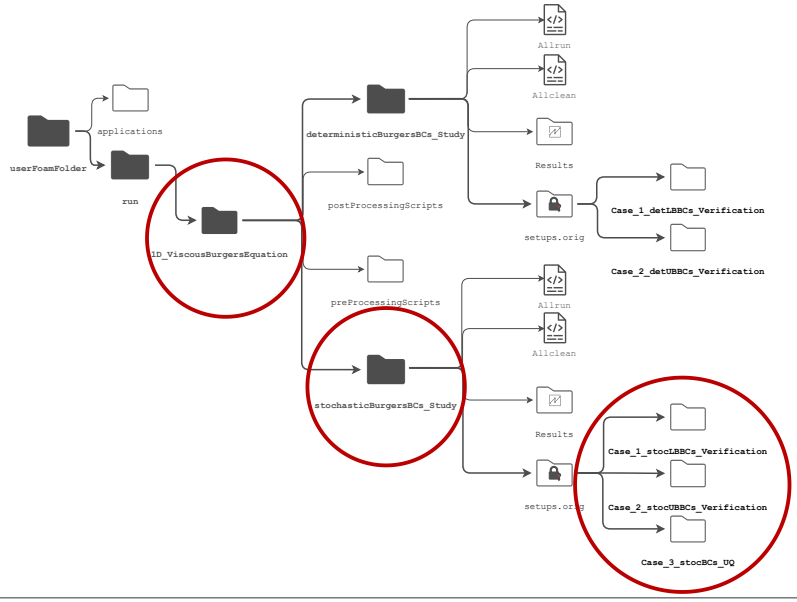
```
def calculate_triple_product(order, distribution,
    polynomials):
    e_ijk = {}
    num_polynomials = len(polynomials)
    c = 1 # Classical Gauss Quadrature (c=1)
    quadrature_order = math.ceil((3 * order + c)/2)
    nodes, weights = cp.generate_quadrature(
    quadrature_order,

    distribution, rule = "gaussian")
    for i in range(num_polynomials):
        for j in range(num_polynomials):
            for k in range(num_polynomials):
                # Using Quadrature
                integrand = (polynomials[i](nodes)
                             * polynomials[j](nodes)
                             * polynomials[k](nodes))
                e_ijk[(i, j, k)] = np.sum(weights *
    integrand)
    return e_ijk
```

Function Overview

- Computes triple product e_{ijk} for the *Galerkin* tensor in polynomial chaos expansions.
- Uses **Gaussian quadrature**.

Verification & Uncertainty Quantification


userFoamFolder
applications
run
1D_ViscousBurgersEquation
deterministicBurgersBCs_Study
postProcessingScripts
preProcessingScripts
stochasticBurgersBCs_Study
Allrun
Allclean
Results
setups.orig
Case_1_detLBBCs_Verification
Case_2_detUBBCs_Verification
Allrun
Allclean
Results
setups.orig
Case_1_stocLBBCs_Verification
Case_2_stocUBBCs_Verification
Case_3_stocBCs_UQ

Configuration of Stochastic Cases

Configuration: `0.orig (Uhat0)`

```
a  $lower; // a 0.0;
b  $upper; // b 0.1;
Ux_LBC $Ux_LBC;   // Specify a fixed velocity at x = -1
Ux_RBC $Ux_RBC;  // Specify a fixed velocity at x = 1
deltaMean #calc "(($a + $b)/2.0)"; // Uncomment for a real
     stochastic solver.
boundaryField
{
    left
    {
        type            fixedValue;
        value           uniform (#calc "($Ux_LBC + $deltaMean)
      " 0.0 0.0);
    }
    right
    {
        type            fixedValue;
        value           uniform ($Ux_RBC 0.0 0.0);
    }
    other
    {
        type            empty;  // Ignore y and z directions
      for 1D simulation
    }
}
```

Configuration Overview

- `Case_1_stocLBBCs_Verification`
  → `deltaMean $a;`
- `Case_2_stocUBBCs_Verification`
  → `deltaMean $b;`
- `Case_3_stocBCs_UQ`
  → `deltaMean #calc "(($a + $b)/2.0)";`

Configuration of Stochastic Cases

Configuration: `0.orig (Uhat1)`

```
#include "../constant/UQProperties"
a  $lower; // a 0.0;
b  $upper; // b 0.1;
deltaSigma #calc "($b - $a) / (2.0 * sqrt(3.0))";
boundaryField
{
    left
    {
        type            fixedValue;
        value           uniform ($deltaSigma 0.0 0.0);
    }
    right
    {
        type            fixedValue;
        value           uniform (0.0 0.0 0.0);
    }
    other
    {
        type            empty;
    }
}
```

Configuration Overview

- `Case_1_stocLBBCs_Verification`
  → `deltaMean 0;`
- `Case_2_stocUBBCs_Verification`
  → `deltaMean 0;`
- `Case_3_stocBCs_UQ`
  → `deltaSigma #calc "($b - $a) / (2.0 * sqrt(3.0))";`

Execution of Stochastic Cases

Automation Script

Streamlines stochastic case execution using `Allclean` and `Allrun` in OpenFOAM (v2306). Requires `pyFoam` and `chaospy`.

`Allrun` Script

Automates pre- and post-processing tasks

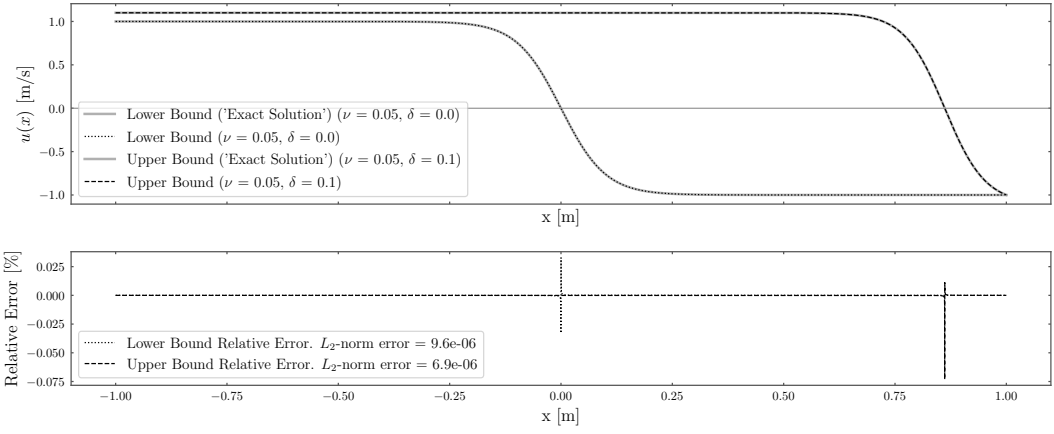
- Prepares directories, generates and verifies mesh (`blockMesh`, `checkMesh`).
- Supports serial and parallel execution.
- Collects results, converts to VTK format, and runs Python post-processing.
- Cleans temporary files after each case.

`Allclean` Script

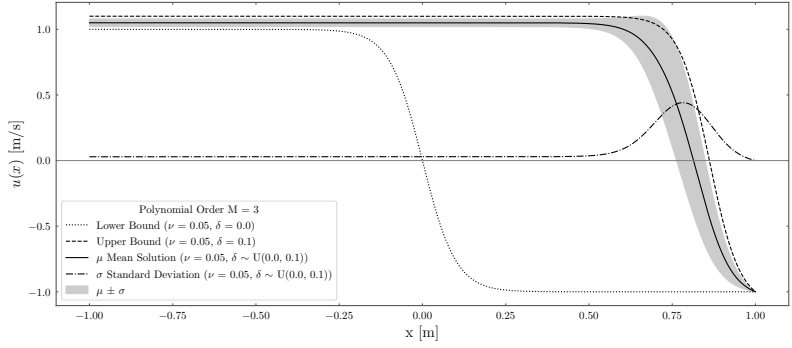
Cleans working directory by removing:

- Temporary files, simulation results, sampled data.
- Case setups (`0.orig`, `constant`, `system`) and VTK files.

Ensures a clean structure for future simulations.



δ	z_{ex}	z_{stoch}
0	0.0000000**0**	-0.0000000**3**
10^{-1}	0.8616126**2**	0.8616126**0**



Polynomial Order (P)	$\overline{z}_{ref}$ [8]	$\overline{z}$	σ_{zref}	σ_z
$P = 1$	0.814592**94**	0.814592**91**	0.376607**51**	0.376607**41**
$P = 2$	0.813940**90**	0.813940**87**	0.410993**50**	0.410993**33**
$P = 3$	0.813906**71**	0.813906**68**	0.413820**35**	0.413820**14**

[8] Reference values ($\overline{z}_{ref}$, σ_{zref}) documented in Table B1 of Ref. [12].

Conclusions & Future Work

Conclusions

- Tutorial for using OpenFOAM in intrusive generalised polynomial chaos frameworks has been provided.
- Implementation modifications and verification of `burgersFoam` and `gPCBurgersFoam`.
- Verified solvers for UQ in the 1D viscous Burgers' equation with high accuracy.
- Supports improvement and expansion of UQ in CFD using OpenFOAM.

Future Work

- **Polynomial Chaos Methods.** Extend to Hermite polynomials for broader UQ applications.
- **Complex Problems.** Test with higher dimensions and advanced equations.
- **Hydrogen Deflagration.** Apply framework to hydrogen combustion with turbulence uncertainties.

Relevant References I

[1] George EP Box and Norman R Draper. *Empirical model-building and response surfaces*. John Wiley & Sons, 1987. isbn: 0471810339.

[2] S. Schlesinger. "Terminology for model credibility". In: *SIMULATION* 32.3 (1979), pp. 103–104. issn: 0037-5497. doi: 10.1177/003754977903200304. url: https://doi.org/10.1177/003754977903200304.

[3] SUSANA-Project. *Report on Verification and Validation Procedures*. Report SUSANA Final Report D4.2 December 2016.p. EU Framework Program, 2016. url: https://www.h2fc-net.eu/app/download/10705642983/SUSANA+Final+Report+D4.2+December+2016.pdf?t=1555511013.

[4] ASME. *Standard for Verification and Validation in Computational Fluid Dynamics and Heat Transfer V V 20 - 2009(R2021)*. Standard. ASME (American Society of Mechanical Engineers). 2009. url: https://www.asme.org/codes-standards/find-codes-standards/v-v-20-standard-verification-validation-computational-fluid-dynamics-heat-transfer/2009/print-book.

[5] ASME. *Verification, Validation, and Uncertainty Quantification Terminology in Computational Modeling and Simulation VVUQ 1 - 2022*. Standard. ISBN: 9780791875506. 2022, p. 24. url: https://www.asme.org/codes-standards/find-codes-standards.

[6] Patrick J Roache. *Fundamentals of verification and validation*. hermosa publ., 2009. isbn: 0913478121. url: https://fenix.tecnico.ulisboa.pt/downloadFile/281536824240039O/FVV_Roache_2009.pdf.

[7] James Warner. *Mini Tutorial 6: An Introduction to Uncertainty Quantification for Modeling & Simulation*. 2023. url: https://youtu.be/7w-K_EF2j64?t=418.

[8] Ryan G. McClarren. *Uncertainty Quantification and Predictive Computational Science. A Foundation for Physical Scientists and Engineers*. Springer Cham, 2018. isbn: 978-3-319-99525-0. doi: https://doi-org.ludwig.lub.lu.se/10.1007/978-3-319-99525-0. url: https://link-springer-com.ludwig.lub.lu.se/book/10.1007/978-3-319-99525-0.

[9] T.J. Sullivan. *Introduction to Uncertainty Quantification*. Springer, 2015. isbn: 978-3-319-23395-6. doi: https://doi.org/10.1007/978-3-319-23395-6. url: https://link.springer.com/book/10.1007/978-3-319-23395-6.

[10] Svenska Institutet för Standarder and SS-EN ISO 13943:2017. *Fire safety – Vocabulary (ISO 13943:2017)*. Standard. 2017. url: https://www.sis.se/produkter/terminologi-och-dokumentation/ordlistor/miljo-och-halsoskydd/ss-en-iso-139432017/.

[11] A. R. Masri et al. "A Comparative Study of Turbulent Premixed Flames Propagating Past Repeated Obstacles". In: *Industrial & Engineering Chemistry Research* 51.22 (2012). doi: 10.1021/ie201928g, pp. 7690–7703. issn: 0888-5885. doi: 10.1021/ie201928g. url: https://doi.org/10.1021/ie201928g.

[12] Dongbin Xiu and George Em Karniadakis. "Supersensitivity due to uncertain boundary conditions". In: *International Journal for Numerical Methods in Engineering* 61.12 (2004). issn: 0029-5981. doi: https://doi.org/10.1002/nme.1152. url: https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1152.

[13] M. P. Bonkile et al. "A systematic literature review of Burgers' equation with recent advances". In: *Pramana - Journal of Physics* 90.6 (2018). issn: 03044289 (ISSN). doi: 10.1007/s12043-018-1559-4. url: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85046302650&doi=10.1007%2fs12043-018-1559-4&partnerID=40&md5=c29acd331cea3278444820cc9810b848.

[14] *CFD with OpenSource Software*. http://www.tfd.chalmers.se/~hani/kurser/OS_CFD/. Accessed: 2024-01-20.

[15] R. Ghanem, H. Owhadi, and D. Higdon. *Handbook of uncertainty quantification*. Handbook of Uncertainty Quantification. Export Date: 16 May 2023; Cited By: 58. Springer International Publishing, 2017, pp. 1–2053. isbn: 978-331912385-1 (ISBN); 978-331912384-4 (ISBN). doi: 10.1007/978-3-319-12385-1. url: https://www.scopus.com/inward/record.uri?eid=2-s2.0-85020310085&doi=10.1007%2f978-3-319-12385-1&partnerID=40&md5=0fd5931878488a2bb56f1c7fcb55dfd6.

[16] *PyFoam 2023.7*. Accessed: 2024-11-18. url: https://pypi.org/project/PyFoam/.

[17] *Chaospy — chaospy 4.3.13 documentation*. Accessed: 2024-11-18. url: https://chaospy.readthedocs.io/en/master/.

"When you can measure what you are speaking about and express it in numbers you know something about it; but when you can not express it in numbers your knowledge is of a meagre and unsatisfactory kind."

Lord Kelvin, British Scientist(1824-1907)

LUND
UNIVERSITY

LTH

FACULTY OF
ENGINEERING

Report, Presentation and Code.