

Determining the Model Coefficients of a New Turbulence Model

DAVID J. TOBIN

University of Wyoming

Department of Mechanical Engineering

2018-2019 Wyoming NASA Space Grant Consortium

Undergraduate Research Fellowship

Abstract

Turbulent flow in a fluid is flow characterized by random, irregular, and fluctuating motion. The ability to model such chaotic flows is necessary for countless applications, many of which are of great interest to NASA (e.g. predicting stalls in fixed-wing flight, calculating turbulent effects on the efficiencies in gas turbine engines, designing for turbulent mixing in internal combustion engines, etc.) Exact numerical simulations of such flows are possible but require repetitively solving the three-dimensional Navier-Stokes equations to obtain the fluid flow field, and then averaging the solutions to obtain applicable statistics.¹ Such an approach requires an extremely demanding amount of computational power and thus is not a practical solution for many common flows of engineering interest. Therefore, the goal of turbulence theories and models is to describe turbulent motion by exact analytical methods.¹ Richard Feynman denoted this task as the last great unsolved problem of classical physics.

In 1991, J.L. Lumley proposed a new model² for calculating the energy dissipation rate in a flow (a key parameter for modeling turbulent behavior as a whole.) This new model has the potential to rectify the shortcomings of all currently adopted turbulence models by accounting for the history of the strain rate that fluid elements experience in a turbulent flow, yet, there is no evidence that this proposed model was ever compared to data from a direct numerical simulation (DNS) in order to determine the optimal constants and initial conditions for the new Lumley model. Therefore, the goal of this research was to test and optimize Lumley's proposed turbulence model by referencing DNS data. The "standard" and the "realizable" turbulence models are also used for comparison. Theoretical derivations are provided for each model. Data from eight DNS studies of varying nature are used to assess the performance of each model.

Table of Contents

Introduction	1
Derivations of Studied Models	1
Parameters of Studied Cases	9
Results	14
Conclusions	19
Future Work	20
Acknowledgments	20
Appendix I: Symbols	21
Appendix II: References	22
Appendix III: Python Code	23

List of Figures

Figure 1	C128W Results	15
Figure 2	C128U Results	15
Figure 3	Case2 Results	16
Figure 4	HSh Results	16
Figure 5	Decay Results	17
Figure 6	ω_{Low} Results	17
Figure 7	ω_{Med} Results	18
Figure 8	ω_{High} Results	18

Introduction

The overall objective of this research is to numerically analyze and optimize the model for the energy dissipation rate in a turbulent flow proposed by J.L. Lumley in 1991². The publication of this model included a complete theoretical derivation, but it did not include any evidence as to whether the model is consistent with turbulent flow data from a direct numerical simulation (DNS). This was likely due to a lack of computational resources at the time. In 1994, Shih et al.³ published a simplified version of Lumleys proposed model that was much more reasonable to compute. However, despite the improved capabilities of modern computational tools, there exists no evidence that Lumleys original model was ever reconsidered. Therefore, this research will utilize modern DNS data to assess in which cases the proposed Lumley model is able to accurately predict the behavior of turbulent kinetic energy in a flow with respect to time, and to ultimately determine what constants and initial conditions are necessary for it to do so.

Derivations of Studied Models

Six turbulence models are compared to DNS data for each considered flow case. These models are defined as the standard $k - \varepsilon$ model, the realizable $k - \varepsilon$ model, the Lumley $k - \varepsilon$ model, the standard Reynolds stress model, the realizable Reynolds stress model, and the Lumley Reynolds stress model. The models are reduced to a set of linear ODE's such that they can be solved with the 4th order Runge-Kutta method, using the initial conditions and parameters described later for each case. Theoretical derivation for these models, and the selection for their respective constants, are provided in the following sections.

Standard $k - \varepsilon$ Model

From Durbin¹ (p. 122), the general transport equation for k is:

$$\frac{\partial k}{\partial t} + U_j \frac{\partial k}{\partial x_j} = \mathcal{P} - \varepsilon + \frac{\partial((\nu + \nu_t) \frac{\partial k}{\partial x_j})}{\partial x_j} \quad (1)$$

For homogeneous flow, all spatial derivatives are zero. Therefore this reduces to:

$$\frac{\partial k}{\partial t} = \mathcal{P} - \varepsilon \quad (2)$$

The production term, $\mathcal{P}$, can be related to S_s as follows:

$$\mathcal{P} = -\overline{u_i u_j} \partial_j U_i \quad (\text{by definition}) \quad (3)$$

$$= 2\nu_t S_{ij} \partial_j U_i - \frac{2}{3} k \partial_i U_i \quad (\text{by substituting } -\overline{u_i u_j} = 2\nu_t S_{ij} - \frac{2}{3} k \partial_{ij}) \quad (4)$$

$$= 2\nu_t S_{ij} S_{ij} \quad (\text{assuming incompressible flow}) \quad (5)$$

$$= S_s^2 C_\mu \frac{k^2}{\varepsilon} \quad (\text{assuming the stress-intensity ratio } \frac{\overline{uv}}{k} \approx 0.3 = C_\mu^2 \text{ in the log layer}) \quad (6)$$

Substituting into (2) gives:

$$\boxed{\frac{\partial k}{\partial t} = C_\mu S_s^2 \frac{k^2}{\varepsilon} - \varepsilon} \quad (7)$$

From Durbin¹ (p. 122), the general transport equation for ε is:

$$\frac{\partial \varepsilon}{\partial t} + U_j \frac{\partial \varepsilon}{\partial x_j} = \frac{C_{\varepsilon 1} \mathcal{P} - C_{\varepsilon 2} \varepsilon}{T} + \frac{\partial \left(\left(\nu + \frac{\nu_t}{\sigma_\varepsilon} \right) \frac{\partial \varepsilon}{\partial x_j} \right)}{\partial x_j} \quad (8)$$

where $T = \frac{k}{\varepsilon}$ is the turbulence time scale. The general equation is then simplified as follows:

$$\frac{\partial \varepsilon}{\partial t} = \frac{C_{\varepsilon 1} \mathcal{P} - C_{\varepsilon 2} \varepsilon}{T} \quad (\text{assuming homogeneous flow}) \quad (9)$$

$$\frac{\partial \varepsilon}{\partial t} = \left(C_{\varepsilon 1} \mathcal{P} - C_{\varepsilon 2} \varepsilon \right) \frac{\varepsilon}{k} \quad (\text{substituting T definition}) \quad (10)$$

$$\boxed{\frac{\partial \varepsilon}{\partial t} = \left(C_{\varepsilon 1} C_\mu S_s^2 \frac{k^2}{\varepsilon} - C_{\varepsilon 2} \varepsilon \right) \frac{\varepsilon}{k}} \quad (\text{substituting (7)}) \quad (11)$$

Therefore equations (7) and (11) are the simplified ODE's for the standard $k - \varepsilon$ turbulence model, assuming homogeneous, incompressible, parallel shear flow. The “standard” values for the constants used in these equations are: $C_\mu = 0.09$, $C_{\varepsilon 1} = 1.44$, $C_{\varepsilon 2} = 1.92$.

Realizable $k - \varepsilon$ Model

Shih et al.³ proposed the realizable $k - \varepsilon$ model in 1995. The term “realizable” indicates that certain mathematical constraints on Reynolds stresses are satisfied. The model utilizes the standard

k equation, therefore:

$$\boxed{\frac{\partial k}{\partial t} = C_\mu S_s^2 \frac{k^2}{\varepsilon} - \varepsilon} \quad (12)$$

However, unlike in the standard $k - \varepsilon$ model, C_μ is now made variable by sensitizing it to the mean flow and the turbulence. Reynolds and Shih et al.³ propose the following formulation for C_μ :

$$C_\mu = \frac{1}{A_0 + A_S U^{(*)} \frac{k}{\varepsilon}} \quad (13)$$

$$U^{(*)} = \sqrt{S_{ij}S_{ij} + \tilde{\Omega}_{ij}\tilde{\Omega}_{ij}} \quad (14)$$

$$\tilde{\Omega}_{ij} = \Omega_{ij} - 2\varepsilon_{ijk}\omega_k \quad (15)$$

$$\Omega_{ij} = \bar{\Omega}_{ij} - \varepsilon_{ijk}\omega_k \quad (16)$$

$$A_0 = 4.04 \quad (17)$$

$$A_S = \sqrt{6} \cos \phi \quad (18)$$

$$\phi = \frac{1}{3} \arccos(\sqrt{6}W) \quad (19)$$

$$W = \frac{S_{ij}S_{jk}S_{ki}}{(\sqrt{S_{ij}S_{ij}})^3} \quad (20)$$

For parallel shear flow:

$$W = 0 \quad (21)$$

$$\phi = \frac{\pi}{6} \quad (22)$$

$$A_S = \frac{3\sqrt{2}}{2} \quad (23)$$

$$U^{(*)} = |S_s| \quad (\text{assuming no rotation}) \quad (24)$$

The proposed transport equation for ε is:

$$\frac{\partial \varepsilon}{\partial t} + U_j \frac{\partial \varepsilon}{\partial x_j} = \frac{\partial(-\overline{u_j \varepsilon'})}{\partial x_j} + C_1 \sqrt{2S_{ij}S_{ij}}\varepsilon - C_2 \frac{\varepsilon^2}{k + \sqrt{\nu\varepsilon}} \quad (25)$$

The equation is reduced as follows:

$$\frac{\partial \varepsilon}{\partial t} = C_1 \sqrt{2S_{ij}S_{ij}}\varepsilon - C_2 \frac{\varepsilon^2}{k + \sqrt{\nu\varepsilon}} \quad (\text{assuming homogeneous flow}) \quad (26)$$

$$\frac{\partial \varepsilon}{\partial t} = C_1 |S_s|\varepsilon - C_2 \frac{\varepsilon^2}{k + \sqrt{\nu\varepsilon}} \quad (27)$$

$$\boxed{\frac{\partial \varepsilon}{\partial t} = C_1 |S_s|\varepsilon - C_2 \frac{\varepsilon^2}{k}} \quad (\text{neglecting viscosity}) \quad (28)$$

The new constants C_1 and C_2 are proposed to be:

$$C_1 = \max \left\{ 0.43, \frac{\eta}{5 + \eta} \right\} \quad (29)$$

$$\eta = |S_s| \frac{k}{\varepsilon} \quad (30)$$

$$C_2 = \frac{n + 1}{n} \quad (31)$$

$$n = 1.11 \quad (32)$$

Lumley $k - \varepsilon$ Model

The new Lumley $k - \varepsilon$ model is defined as the realizable $k - \varepsilon$ model with the addition of a new rate quantity, $\mathcal{S}$, defined by the following equation from Lumley² (1992):

$$\frac{\partial \mathcal{S}}{\partial t} + U_j \frac{\partial \mathcal{S}}{\partial x_j} = \frac{\partial}{\partial x_j} \left(\nu_t \frac{\partial \mathcal{S}}{\partial x_j} \right) + \left(\sqrt{S_{ij}S_{ij}} - \mathcal{S} \right) C_{S1} \frac{\varepsilon}{k} \quad (33)$$

This equation can be reduced as follows:

$$\frac{\partial \mathcal{S}}{\partial t} = \left(\sqrt{S_{ij}S_{ij}} - \mathcal{S} \right) C_{S1} \frac{\varepsilon}{k} \quad (\text{assuming homogeneous flow}) \quad (34)$$

$$\boxed{\frac{\partial \mathcal{S}}{\partial t} = \left(\frac{|S_s|}{\sqrt{2}} - \mathcal{S} \right) C_{S1} \frac{\varepsilon}{k}} \quad (35)$$

This new parameter $\mathcal{S}$ is then coupled with the ε equation from the realizable $k - \varepsilon$ model in the following way:

$$\boxed{\frac{\partial \varepsilon}{\partial t} = \sqrt{2} C_1 \mathcal{S} \varepsilon - C_2 \frac{\varepsilon^2}{k}} \quad (36)$$

such that $\mathcal{S}$ approaches $\frac{S_s}{\sqrt{2}}$ and thus the Lumley model approaches the realizable $k - \varepsilon$ model after a sufficient amount of time.

The k equation from the realizable $k - \varepsilon$ model is also still used here:

$$\boxed{\frac{\partial k}{\partial t} = C_\mu S_s^2 \frac{k^2}{\varepsilon} - \varepsilon} \quad (37)$$

The constants C_μ and C_1 remain as they are described in the realizable $k - \varepsilon$ model (i.e. equations (13) – (20) and (29) – (30)). C_2 however is now made variable by $C_2 = \frac{n+1}{n} + C_1$. Therefore for $n = 1.11$, $C_2 = 1.9 + C_1$.

The new constant, C_{S1} is determined by first assuming the following asymptotic relationships:

$$k = k_0 \left(\frac{t}{t_0} \right)^{-n} \quad (38)$$

$$\varepsilon = \varepsilon_0 \left(\frac{t}{t_0} \right)^{-(n+1)} \quad (39)$$

$$\mathcal{S} = \mathcal{S}_0 \left(\frac{t}{t_0} \right)^{-1} \quad (40)$$

For decaying turbulence, the new Lumley model gives $\frac{\partial \mathcal{S}}{\partial t} = -C_{S1} \frac{\varepsilon}{k} \mathcal{S}$. Substituting the asymptotic relationships into this form gives:

$$-\mathcal{S}_0 \frac{t_0}{t^2} = -C_{S1} \frac{\varepsilon}{k} \mathcal{S}_0 \left(\frac{t}{t_0} \right)^{-1} \quad (41)$$

$$\frac{1}{t} = C_{S1} \frac{\varepsilon_0 \left(\frac{t}{t_0} \right)^{-(n+1)}}{k_0 \left(\frac{t}{t_0} \right)^{-n}} \quad (42)$$

$$\frac{1}{t} = C_{S1} \frac{\varepsilon_0}{k_0} \left(\frac{t}{t_0} \right)^{-1} \quad (43)$$

$$\frac{1}{t_0} = C_{S1} \frac{n k_0}{t_0 k_0} \quad \left(\text{because } \varepsilon_0 = \frac{n k_0}{t_0} \right) \quad (44)$$

$$\boxed{C_{S1} = \frac{1}{n}} \quad (45)$$

Therefore for $n = 1.11$, $C_{S1} = 0.9$.

Standard Reynolds Stress Model (RSM_S)

The general Reynolds stress model transport equation⁴ (assuming no diffusion and no production by body forces) is:

$$\frac{\partial \overline{u_i u_j}}{\partial t} = P_{ij} - \varepsilon_{ij} + \Phi_{ij} \quad (46)$$

$$P_{ij} = - \left(\overline{u_i u_k} \frac{\partial u_j}{\partial x_k} + \overline{u_j u_k} \frac{\partial u_i}{\partial x_k} \right) \quad (47)$$

$$\varepsilon_{ij} = \frac{2}{3} \varepsilon \delta_{ij} \quad (48)$$

$$\Phi_{ij} = \Phi_{ij1} + \Phi_{ij2} \quad (49)$$

$$\Phi_{ij1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_i u_j} - \frac{2}{3} \delta_{ij} k \right) \quad (50)$$

$$\Phi_{ij2} = -C_{R2} \left(P_{ij} - \frac{2}{3} \delta_{ij} P_k \right) \quad (51)$$

$$P_k = -\overline{u_i u_k} \frac{\partial u_i}{\partial x_k} \quad (52)$$

with $C_{R1} = 1.8$ and $C_{R2} = 0.6$. The turbulent kinetic energy k is then calculated as:

$$k = \frac{1}{2} [\overline{u_1 u_1} + \overline{u_2 u_2} + \overline{u_3 u_3}] \quad (53)$$

Therefore, equations (46) – (52) must be computed for the four cases $i = j = 1$, $i = j = 2$, $i = j = 3$, and $i = 1 \quad j = 2$.

$i = 1 \quad j = 1$:

$$P_{11} = -2(\overline{u_1 u_2} S_s) \quad (54)$$

$$\varepsilon_{11} = \frac{2}{3} \varepsilon \quad (55)$$

$$\Phi_{111} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_1} - \frac{2}{3} k \right) \quad (56)$$

$$\Phi_{112} = C_{R2} \frac{4}{3} \overline{u_1 u_2} S_s \quad (57)$$

$$\Phi_{11} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_1} - \frac{2}{3} k \right) + C_{R2} \frac{4}{3} \overline{u_1 u_2} S_s \quad (58)$$

$$\frac{\partial \overline{u_1 u_1}}{\partial t} = -2 \left(\overline{u_1 u_2} S_s \right) - \frac{2}{3} \varepsilon - C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_1} - \frac{2}{3} k \right) + C_{R2} \frac{4}{3} \overline{u_1 u_2} S_s \quad (59)$$

$i = 2 \quad j = 2 :$

$$P_{22} = 0 \quad (60)$$

$$\varepsilon_{22} = \frac{2}{3}\varepsilon \quad (61)$$

$$\Phi_{22_1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_2 u_2} - \frac{2}{3}k \right) \quad (62)$$

$$\Phi_{22_2} = -C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s \quad (63)$$

$$\Phi_{22} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_2 u_2} - \frac{2}{3}k \right) - C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s \quad (64)$$

$$\boxed{\frac{\partial \overline{u_2 u_2}}{\partial t} = -\frac{2}{3}\varepsilon - C_{R1} \frac{\varepsilon}{k} \left(\overline{u_2 u_2} - \frac{2}{3}k \right) - C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s} \quad (65)$$

$i = 3 \quad j = 3 :$

$$P_{33} = 0 \quad (66)$$

$$\varepsilon_{33} = \frac{2}{3}\varepsilon \quad (67)$$

$$\Phi_{33_1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_3 u_3} - \frac{2}{3}k \right) \quad (68)$$

$$\Phi_{33_2} = -C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s \quad (69)$$

$$\Phi_{33} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_3 u_3} - \frac{2}{3}k \right) - C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s \quad (70)$$

$$\boxed{\frac{\partial \overline{u_3 u_3}}{\partial t} = -\frac{2}{3}\varepsilon - C_{R1} \frac{\varepsilon}{k} \left(\overline{u_3 u_3} - \frac{2}{3}k \right) - C_{R2} \frac{2}{3} \overline{u_1 u_2} S_s} \quad (71)$$

$i = 1 \quad j = 2 :$

$$P_{12} = -\overline{u_2 u_2} S_s \quad (72)$$

$$\varepsilon_{12} = 0 \quad (73)$$

$$\Phi_{12_1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_2} \right) \quad (74)$$

$$\Phi_{12_2} = C_{R2} \overline{u_2 u_2} S_s \quad (75)$$

$$\Phi_{12} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_2} \right) + C_{R2} \overline{u_2 u_2} S_s \quad (76)$$

$$\boxed{\frac{\partial \overline{u_1 u_2}}{\partial t} = -\overline{u_2 u_2} S_s - C_{R1} \frac{\varepsilon}{k} \overline{u_1 u_2} + C_{R2} \overline{u_2 u_2} S_s} \quad (77)$$

Finally, the standard Reynolds stress model will utilize the standard ε equation:

$$\boxed{\frac{\partial \varepsilon}{\partial t} = -\overline{u_1 u_2} S_s \frac{\varepsilon}{k} C_{\varepsilon 1} - C_{\varepsilon 2} \frac{\varepsilon^2}{k}} \quad (78)$$

with $C_{\varepsilon 1}$ and $C_{\varepsilon 2}$ still taking the standard values $C_{\varepsilon 1} = 1.44$ and $C_{\varepsilon 2} = 1.92$.

Realizable Reynolds Stress Model (RSM_R)

The realizable Reynolds stress model will utilize the same formulation for k as the standard Reynolds stress model (i.e. equations (53, 59, 65, 71, 77) with $C_{R1} = 1.8$ and $C_{R2} = 0.6$), but instead of the standard ε model, it will utilize the realizable ε equation:

$$\boxed{\frac{\partial \varepsilon}{\partial t} = C_1 |S_s| \varepsilon - C_2 \frac{\varepsilon^2}{k}} \quad (79)$$

where C_1 and C_2 remain as they are defined in equations (29) – (32).

Lumley Reynolds Stress Model (RSM_L)

The Lumley Reynolds stress model will also utilize the same formulation for k as the standard Reynolds stress model (i.e. equations (53, 59, 65, 71, 77) with $C_{R1} = 1.8$ and $C_{R2} = 0.6$), but instead of the standard ε model, it will utilize the Lumley ε equation:

$$\boxed{\frac{\partial \varepsilon}{\partial t} = \sqrt{2} C_1 S \varepsilon - C_2 \frac{\varepsilon^2}{k}} \quad (80)$$

coupled with the new S equation:

$$\boxed{\frac{\partial S}{\partial t} = \left(\frac{|S_s|}{\sqrt{2}} - S \right) C_{S1} \frac{\varepsilon}{k}} \quad (81)$$

The constants C_1 and C_{S1} remain as they were defined in equations (29) – (30) and (45). C_2 is again made variable by $C_2 = \frac{n+1}{n} + C_1$. Therefore for $n = 1.11$, $C_2 = 1.9 + C_1$.

Parameters of Studied Cases

Eight cases of turbulent flow, each with their own DNS data, are considered. The assigned titles, descriptions, and sources for these cases are as follows:

- “Decay” : decaying homogeneous, isotropic flow (Yoffe & McComb - 2018)⁵
- “C128W” : homogeneous constant shear flow (Rogers - 1986)⁶
- “C128U” : homogeneous constant shear flow (Rogers - 1986)⁶
- “Case2” : homogeneous constant shear flow (Matsumoto - 1994)⁷
- “HSh” : homogeneous constant shear flow (Lee - 1990)⁸
- “ ω_{Low} ” : homogeneous oscillating shear flow (Yu & Girimaji - 2006)⁹
- “ ω_{Med} ” : homogeneous oscillating shear flow (Yu & Girimaji - 2006)⁹
- “ ω_{High} ” : homogeneous oscillating shear flow (Yu & Girimaji - 2006)⁹

The parameters, initial conditions, and constraints necessary for applying the derived models to each flow case are defined in the following sections.

“Decay” Case Parameters

From Yoffe & McComb⁵ (2018):

$$Re_{L_0} = 3828.2$$

$$Re_{\lambda_0} = 353.7$$

$$\nu_0 = 0.0002 \quad \left(\frac{m^2}{s} \right)$$

$$\tau_0 = 1.94 \quad (s)$$

$$S_s = 0 \quad \left(\frac{1}{s} \right)$$

$$U_0 = \sqrt{\frac{Re_{L_0} \nu_0}{\tau_0}} = 0.628 \quad \left(\frac{m}{s} \right)$$

Therefore, the initial conditions necessary to solve the models are:

$$\begin{aligned}
k_0 &= \frac{3}{2}(U_0^2) = 0.592 \quad \left(\frac{m^2}{s^2}\right) \\
\varepsilon_0 &= \left(\frac{U_0^2 \sqrt{15}}{\sqrt{\nu_0} Re_{\lambda_0}}\right)^2 = 0.093 \left(\frac{m^2}{s^3}\right) \\
\mathcal{S}_0 &= \frac{1}{\tau_0} = 0.515 \quad (s) \quad (\text{Chosen}) \\
\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 &= \frac{2}{3}(k_0) = 0.395 \quad \left(\frac{m^2}{s^2}\right) \\
\overline{uv}_0 &= 0 \quad \left(\frac{m^2}{s^2}\right)
\end{aligned}$$

“C128W” Case Parameters

Rogers⁶ (1986) provides the following parameters and initial conditions:

$$\begin{aligned}
k_0 &= 14.7 \\
\varepsilon_0 &= 155 \\
S_s &= 56.568 \\
S_s t_0 &= 12
\end{aligned}$$

Therefore:

$$\begin{aligned}
t_0 &= \frac{S_s t_0}{S_s} = 0.212 \\
\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 &= \frac{2}{3}(k_0) = 9.8 \\
\overline{uv}_0 &= 0
\end{aligned}$$

Finally, we choose:

$$\mathcal{S}_0 = S_s = 56.568$$

because $\tau_0 = \frac{k_0}{\varepsilon_0} = 0.095 \ll t_0 = 0.212$.

“C128U” Case Parameters

Rogers⁶ (1986) provides the following parameters and initial conditions:

$$k_0 = 6.38$$

$$\varepsilon_0 = 41$$

$$S_s = 28.284$$

$$S_s t_0 = 8$$

Therefore:

$$t_0 = \frac{S_s t_0}{S_s} = 0.283$$

$$\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = 4.25$$

$$\overline{uv}_0 = 0$$

Finally, we choose:

$$\mathcal{S}_0 = S_s = 28.284$$

because $\tau_0 = \frac{k_0}{\varepsilon_0} = 0.156 \ll t_0 = 0.283$.

“Case2” Case Parameters

Matsumoto⁷ (1994) provides the following parameters and initial conditions:

$$k_0 = 0.252$$

$$\varepsilon_0 = 0.234$$

$$S_s = 10\sqrt{2}$$

$$S_s t_0 = 2$$

Therefore:

$$t_0 = \frac{S_s t_0}{S_s} = 0.141$$

$$\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = .168$$

$$\overline{uv}_0 = 0$$

Finally, we choose:

$$\mathcal{S}_0 = \frac{\varepsilon_0}{k_0} = 0.929$$

because $\tau_0 = \frac{k_0}{\varepsilon_0} = 1.08 \gg t_0 = 0.141$.

“HSh” Case Parameters

Lee⁸ (1990) provides the following constraints:

$$\eta_0 = \frac{k_0}{\varepsilon_0}(S_s) = 16.7624$$

$$Re_{T_0} = 1230$$

Now assume $\nu = 10^{-5} \left(\frac{m^2}{s} \right)$ and choose $k_0 = 1 \left(\frac{m^2}{s^2} \right)$. Then:

$$q_0^2 = 2(k_0) = 2 \left(\frac{m^2}{s^2} \right)$$

$$\varepsilon_0 = \frac{(q_0^2)^2}{\nu(Re_{T_0})} = 325.2 \left(\frac{m^2}{s^3} \right)$$

$$S_s = \eta_0 \frac{\varepsilon_0}{k_0} = 5451 \left(\frac{1}{s} \right)$$

$$\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = .666 \left(\frac{m^2}{s^2} \right)$$

$$\overline{uv}_0 = 0 \left(\frac{m^2}{s^2} \right)$$

Finally, we choose:

$$\mathcal{S}_0 = S_s = 5451 \left(\frac{1}{s} \right)$$

“ ω_{Low} ” Case Parameters

Yu & Girimaji⁹ (2006) provide the following constraints:

$$S_s = S_{max} \sin(\omega t)$$

$$S_0^* = S_{max} \frac{k_0}{\varepsilon_0} = 3.3$$

$$\frac{\omega}{S_{max}} = 0.25$$

$$S_{max} = 0.0006875$$

Arbitrarily choose $k_0 = 1$. Therefore:

$$\begin{aligned}\varepsilon_0 &= \frac{S_{max}k_0}{S_0^*} = 0.000208 \\ \omega &= 0.25(S_{max}) = 0.000172 \\ \overline{uu}_0 &= \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = .666 \quad \left(\frac{m^2}{s^2}\right) \\ \overline{uv}_0 &= 0 \quad \left(\frac{m^2}{s^2}\right)\end{aligned}$$

Finally, we choose:

$$\mathcal{S}_0 = \frac{\varepsilon_0}{k_0} = 0.000208$$

“ ω_{Med} ” Case Parameters

Yu & Girimaji⁹ (2006) provide the following constraints:

$$\begin{aligned}S_s &= S_{max} \sin(\omega t) \\ S_0^* &= S_{max} \frac{k_0}{\varepsilon_0} = 3.3 \\ \frac{\omega}{S_{max}} &= 0.5 \\ S_{max} &= 0.0006875\end{aligned}$$

Arbitrarily choose $k_0 = 1$. Therefore:

$$\begin{aligned}\varepsilon_0 &= \frac{S_{max}k_0}{S_0^*} = 0.000208 \\ \omega &= 0.5(S_{max}) = 0.000344 \\ \overline{uu}_0 &= \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = .666 \quad \left(\frac{m^2}{s^2}\right) \\ \overline{uv}_0 &= 0 \quad \left(\frac{m^2}{s^2}\right)\end{aligned}$$

Finally, we choose:

$$\mathcal{S}_0 = \frac{\varepsilon_0}{k_0} = 0.000208$$

“ ω_{High} ” Case Parameters

Yu & Girimaji⁹ (2006) provide the following constraints:

$$\begin{aligned}S_s &= S_{max} \sin(\omega t) \\S_0^* &= S_{max} \frac{k_0}{\varepsilon_0} = 3.3 \\ \frac{\omega}{S_{max}} &= 1.0 \\ S_{max} &= 0.0006875\end{aligned}$$

Arbitrarily choose $k_0 = 1$. Therefore:

$$\begin{aligned}\varepsilon_0 &= \frac{S_{max} k_0}{S_0^*} = 0.000208 \\ \omega &= (S_{max}) = 0.0006875 \\ \overline{uu}_0 &= \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}(k_0) = .666 \quad \left(\frac{m^2}{s^2}\right) \\ \overline{uv}_0 &= 0 \quad \left(\frac{m^2}{s^2}\right)\end{aligned}$$

Finally, we choose:

$$S_0 = \frac{\varepsilon_0}{k_0} = 0.000208$$

Results

Each model was solved for each flow case using the 4th order Runge-Kutta method. The following plots show the calculated solutions for the turbulent kinetic energy, k , as a function of time. The corresponding DNS data that was digitized for each flow case is also shown on its respective plot. The complete Python code that was used to solve the models and generate the following plots is given in Appendix III.

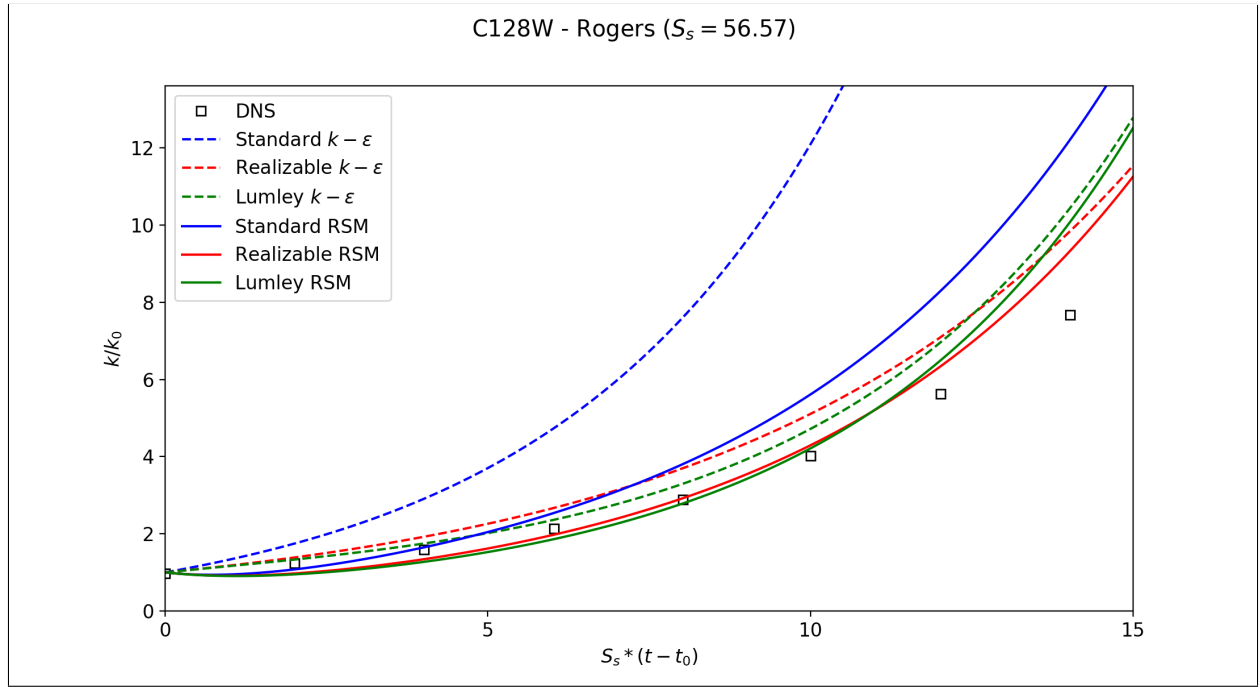


Figure 1: C128W Results

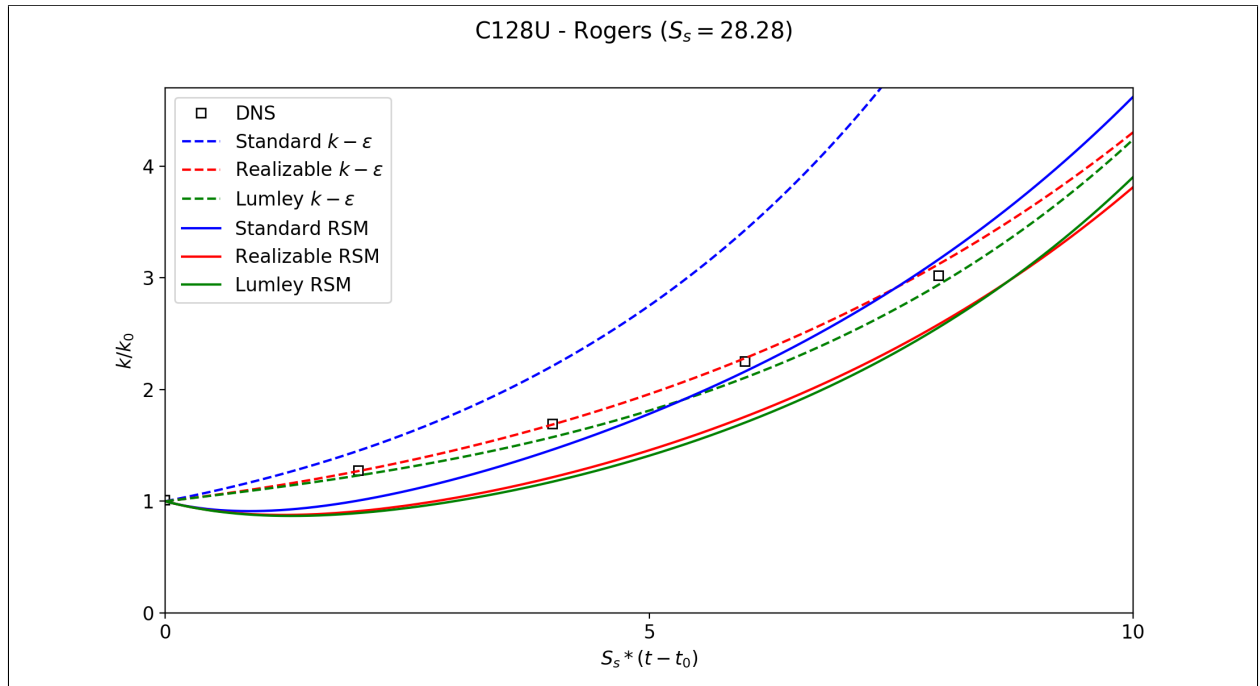


Figure 2: C128U Results

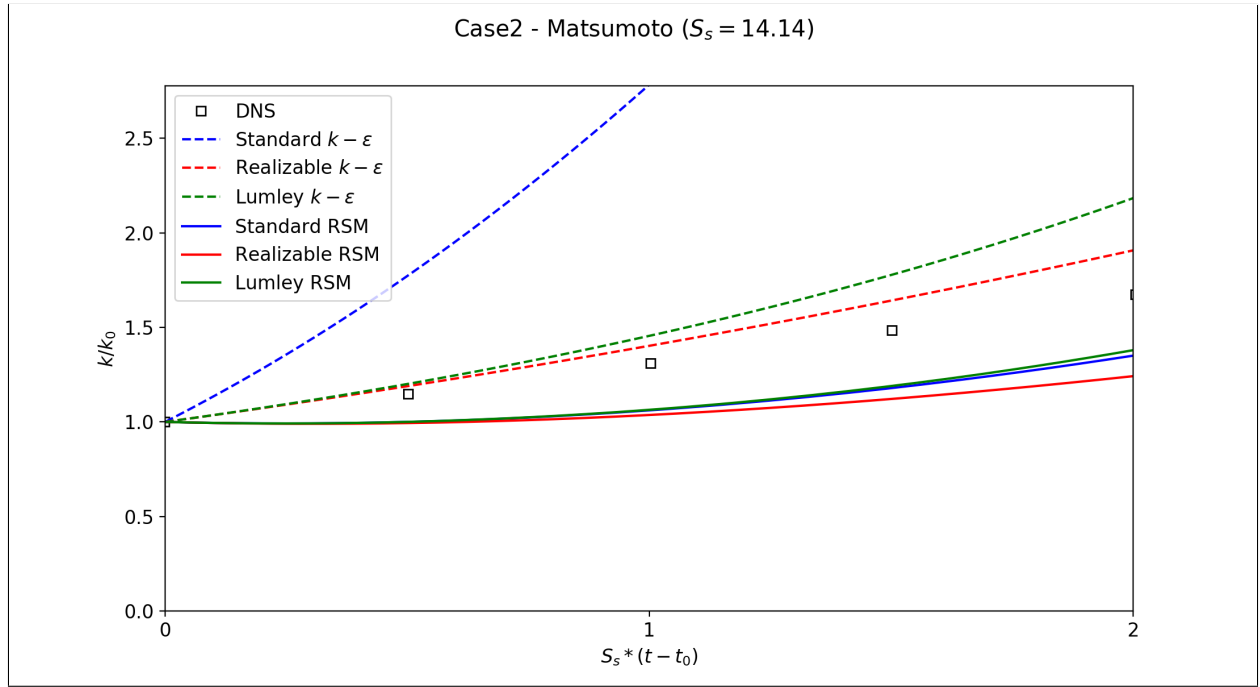


Figure 3: Case2 Results

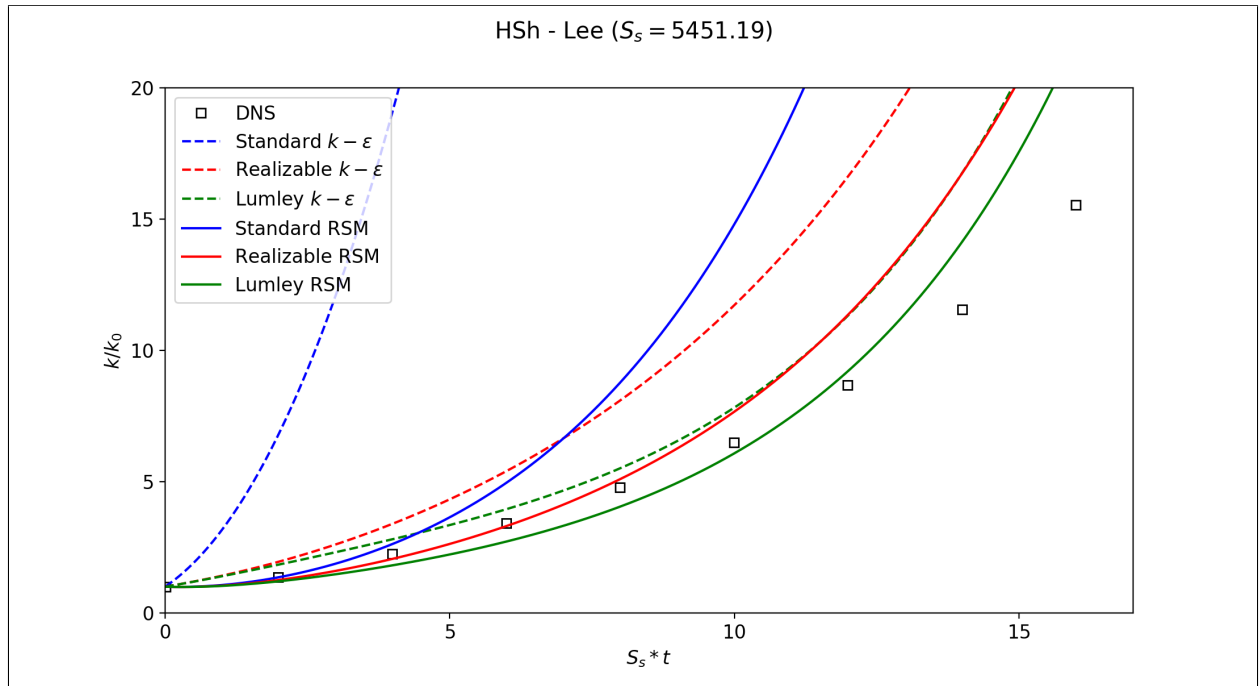


Figure 4: HSh Results

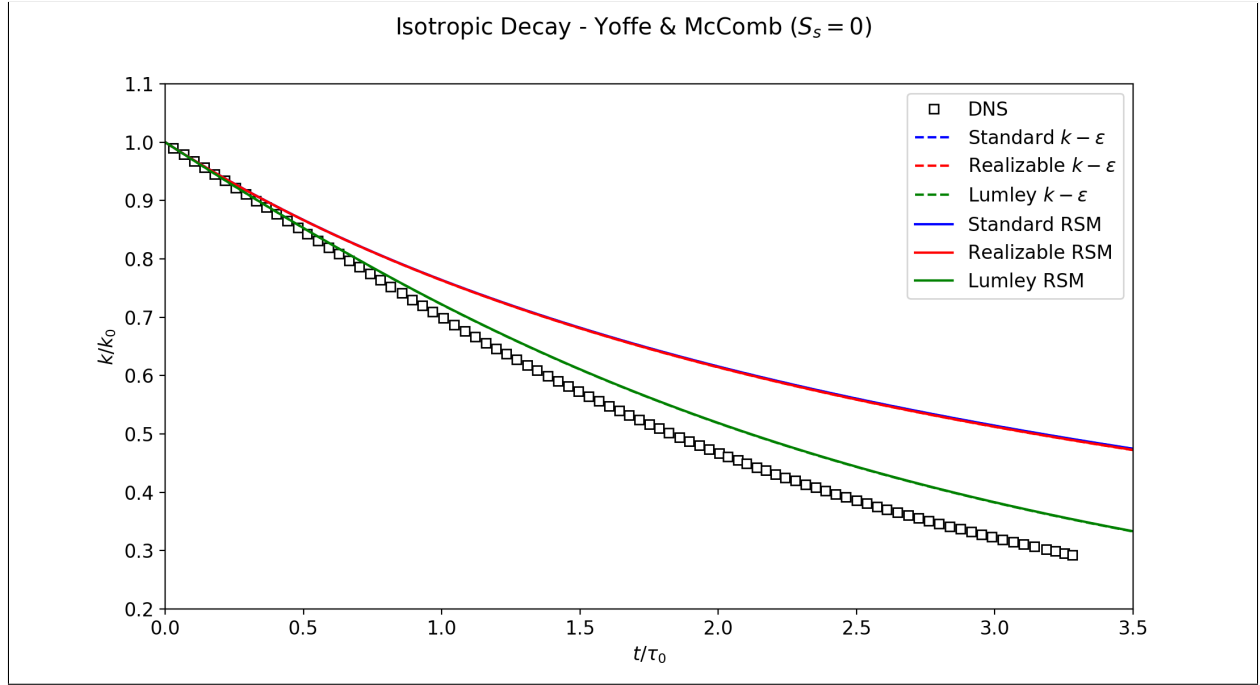


Figure 5: Decay Results

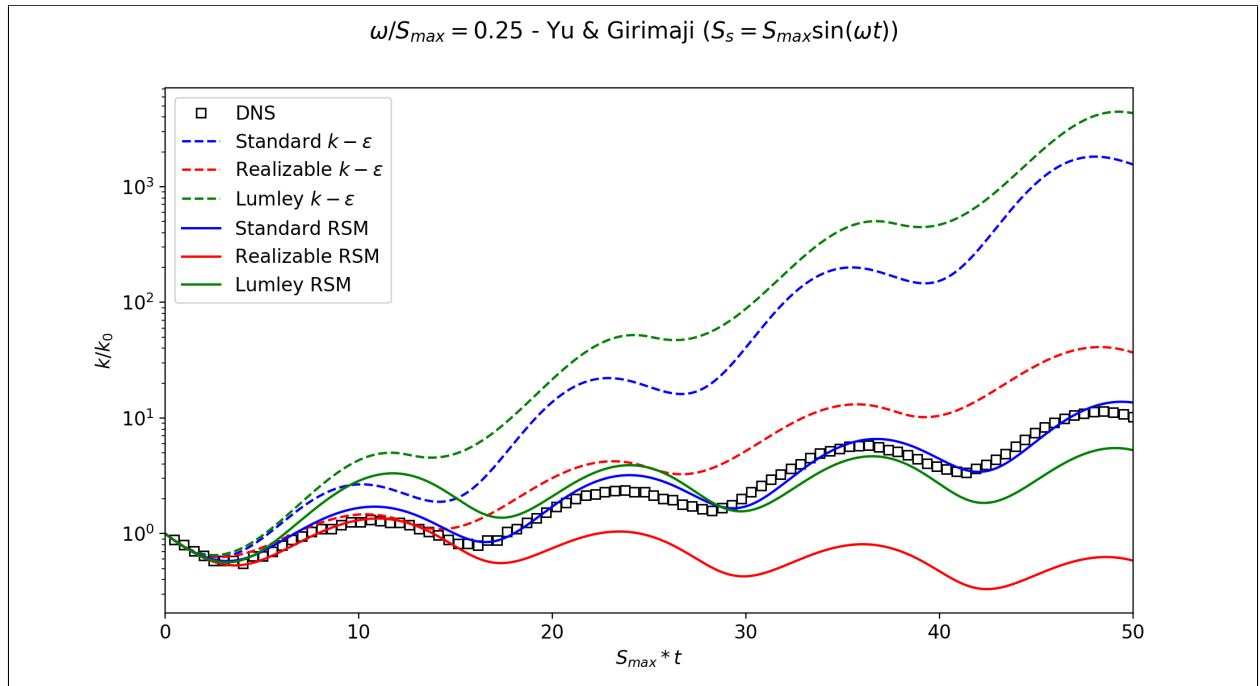


Figure 6: ω_{Low} Results

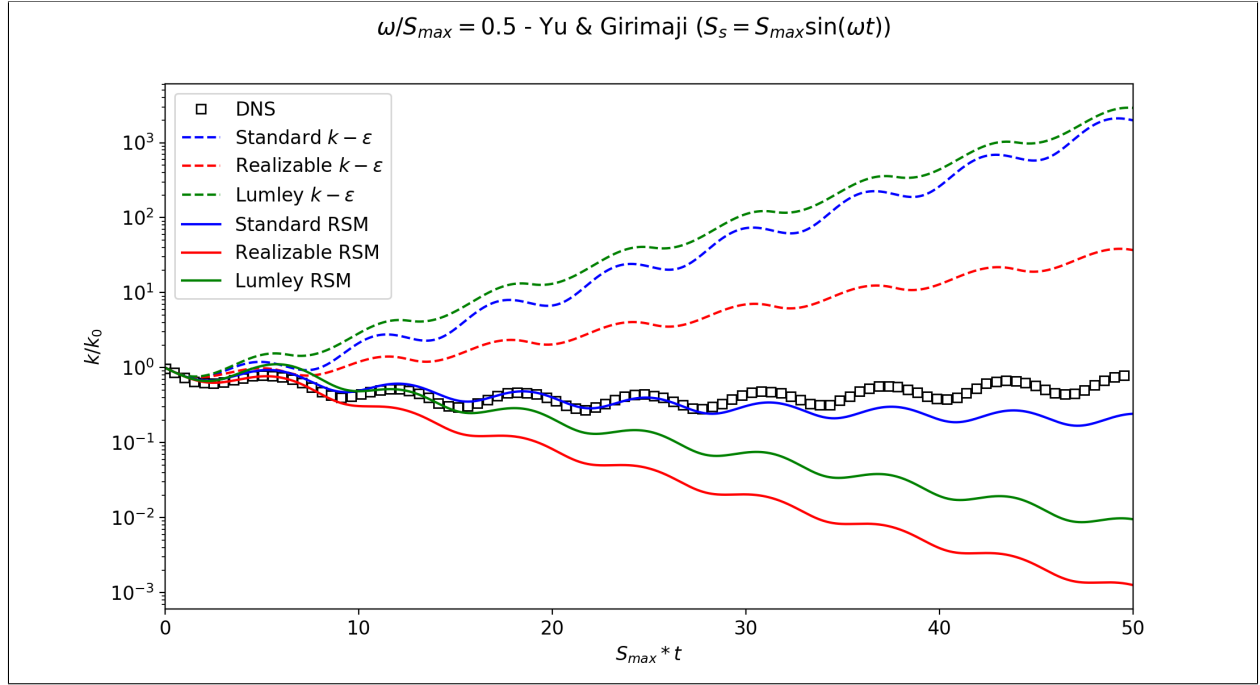


Figure 7: ω_{Med} Results

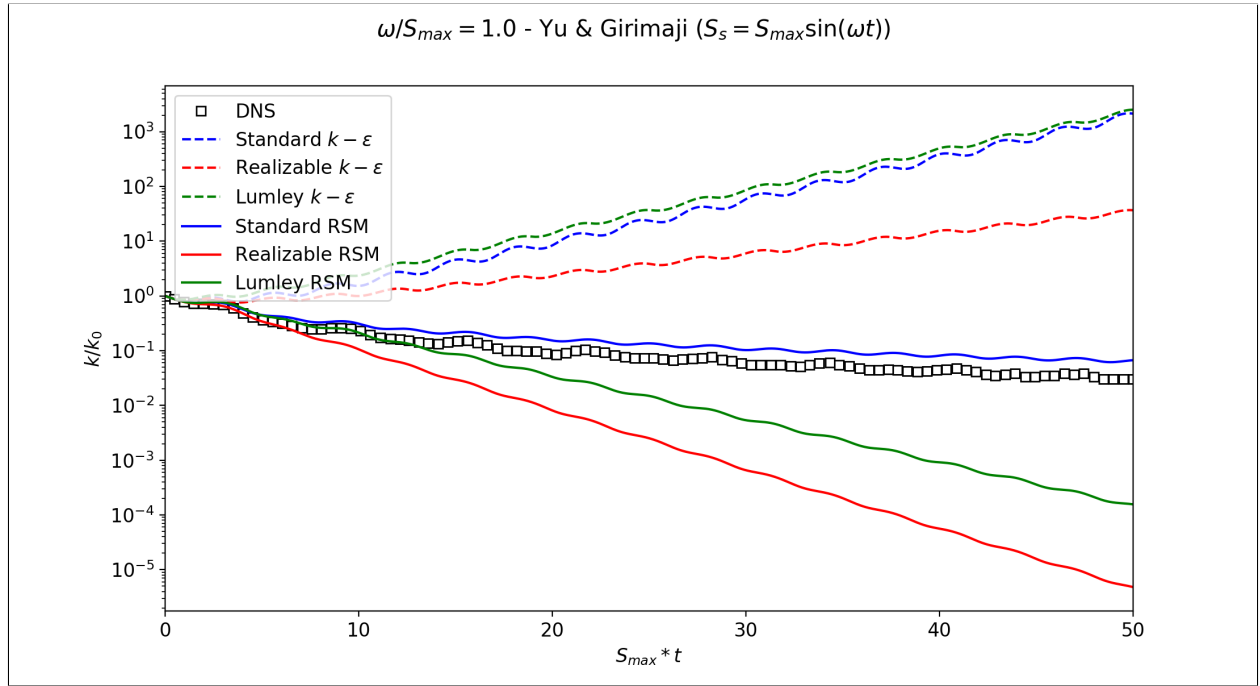


Figure 8: ω_{High} Results

Conclusions

As is typical with modeling turbulent flow, the new Lumley model performs well in certain flow cases, but clearly does not universally outperform the traditional models. **Figures 1-3** show that in cases of low to moderate constant shear, the Lumley model produces results extremely similar to the results from the realizable model (both of which in turn can outperform the standard model, if the appropriate $k - \varepsilon$ vs. RSM selection is used). However in cases of very high shear, as seen in **Figure 4**, the Lumley RSM model greatly outperforms both the standard $k - \varepsilon$ model and the standard RSM model. Furthermore, the Lumley RSM results are similar to the realizable RSM model results early in the simulation, and slightly outperform the realizable RSM model later in the simulation. This result is consistent with the theoretical derivation of the Lumley model, because for a high shear case, the new rate term $\mathcal{S}$ will take a shorter amount of time to reach the equilibrium value (i.e. $\frac{|S_s|}{\sqrt{2}}$). Therefore, for a turbulent flow with a constant shear rate, the Lumley model is more likely to outperform traditional turbulence models if the shear rate is very large.

The results from the isotropic decay case, seen in **Figure 5**, show that in cases with no shear stress, the $k - \varepsilon$ models match the RSM models. Furthermore, the Lumley model greatly outperforms both the standard and the realizable models. This is due to the feature of the Lumley model that makes the C_2 constant a function of the C_1 constant. This result ultimately suggests that in cases of isotropic turbulent decay, the Lumley model is heavily preferable to the traditional models.

Finally, **Figures 6-8** show the results for the periodic shear cases. In general, the Lumley model struggled to produce accurate results in these cases. This is possibly because the oscillating behavior of the shear might conflict with the nature of the Lumley model, which is inherently creating a new rate term that lags behind the shear rate. It is also likely that the initial conditions of the Reynold stress terms would be more suitable if they were derived from the initial condition of the anisotropy tensor (i.e. $\overline{u_i u_{j0}} = (b_{ij0} + \frac{\delta_{ij}}{3}) * 2k$), instead of simply assuming $\overline{u_i u_{j0}} = \frac{2}{3}k$, however the initial anisotropy tensor values were unable to be located from the DNS sources. Further analysis would ultimately be required to determine what changes to the Lumley model constants and/or initial conditions would produce improved results in cases of periodic shear flow.

The new Lumley turbulence model has clearly displayed some merit in modeling the turbulent kinetic energy of a flow, as compared to the traditional “standard” and “realizable” models. Although in its current form it may not universally be the most accurate model to always use, adding it to the list of turbulence models that can carefully and consciously be applied to various types of flows should ultimately aid in the ongoing challenge of modeling turbulent flow.

Future Work

This research effectively exhausted the analysis of the proposed Lumley model that can be done on the level of solving ODE's (ordered differential equations) and comparing the solutions to DNS data. The next step to further validate the model is to implement it into a CFD (computational fluid dynamics) solver such that it can be applied to other cases of turbulent flow. If the proposed model can in fact accurately represent the energy dissipation rate in other cases of turbulent flow, it will continue to prove itself to be an extremely useful tool in the ongoing challenge of modeling turbulent flow. Unfortunately, taking this next step with CFD was not possible during this fellowship due to time constraints. However, the faculty adviser for this project is currently working on this next step, and another University of Wyoming student may even seek another WNSGC undergraduate research fellowship in the future to help continue this work.

Acknowledgments

This research was exclusively funded by the Wyoming NASA Space Grant Consortium through an undergraduate research fellowship. Dr. Michael Stoellinger (University of Wyoming - Department of Mechanical Engineering) served as the sole faculty adviser for this project. This undergraduate research experience was a tremendous learning opportunity for me and it ultimately became the precursor for my ongoing graduate education. Therefore I extend my sincere gratitude for all the support I received along the way.

Appendix I: Symbols

k - turbulent kinetic energy

ε - turbulent dissipation rate

S_s - mean shear rate (i.e. $\sqrt{2S_{ij}S_{ij}}$) Note: For parallel shear flow, $S_s = \frac{\partial U}{\partial y}$

S_{ij} - mean rate of strain tensor $\left(\text{i.e. } \frac{1}{2} \left[\frac{\partial U_j}{\partial x_i} + \frac{\partial U_i}{\partial x_j} \right] \right)$

n - Turbulent decay exponent (experimentally derived)

S - New rate parameter introduced in Lumley model

u - instantaneous velocity

U - mean velocity

ν - viscosity

ν_t - turbulent viscosity

x - position

$\mathcal{P}$ - turbulent production

t - time

τ - turbulent time scale

Re - Reynolds number

ω - sinusoidal frequency

Appendix II: References

1. Durbin, Paul A., and B. A. Petterson. Reif. Statistical Theory and Modeling for Turbulent Flows. Wiley, 2011.
2. Lumley, J. L. Some Comments on Turbulence. *Physics of Fluids A: Fluid Dynamics*, vol. 4, no. 2, 1992, pp. 203211., doi:10.1063/1.858347.
3. Shih, Tsan-Hsing, et al. A New k- Eddy Viscosity Model for High Reynolds Number Turbulent Flows. *Computers & Fluids*, vol. 24, no. 3, 1995, pp. 227238., doi:10.1016/0045-7930(94)00032-t.
4. Klein, T., Craft, T., & Iacovides, H. (2015). Assessment of the performance of different classes of turbulence models in a wide range of non-equilibrium flows. *International Journal of Heat and Fluid Flow*, 51, 229-256. doi:10.1016/j.ijheatfluidflow.2014.10.017
5. Yoffe, S. R., & McComb, W. D. (2018). Onset criteria for freely decaying isotropic turbulence. *Physical Review Fluids*, 3(10). doi:10.1103/physrevfluids.3.104605
6. Rogers, M. M., & Moin, P. (1987). The structure of the vorticity field in homogeneous turbulent flows. *Journal of Fluid Mechanics*, 176(-1), 33. doi:10.1017/s0022112087000569
7. Nagano, Y., Kondoh, M., & Shimada, M. (1997). Multiple time-scale turbulence model for wall and homogeneous shear flows based on direct numerical simulations. *International Journal of Heat and Fluid Flow*, 18(4), 346-359. doi:10.1016/s0142-727x(97)00015-5
8. Lee, M. J., Kim, J., & Moin, P. (1990). Structure of turbulence at high shear rate. *Journal of Fluid Mechanics*, 216(-1), 561. doi:10.1017/s0022112090000532
9. Yu, D., & Girimaji, S. S. (2006). Direct numerical simulations of homogeneous turbulence subject to periodic shear. *Journal of Fluid Mechanics*, 566, 117. doi:10.1017/s0022112006001832

Appendix III: Python Code

```
1 #!/usr/bin/env python
2
3 #Homogeneous Shear Flow Turbulence Model Comparison
4 #Models Compared: Standard k-eps, Realizable k-eps, Lumley k-eps, Standard RSM, Realizable RSM, Lumley RSM
5 #Cases Considered: C128W (Rogers), C128U (Rogers), Case2 (Matsumoto), HSh (Lee), Decay (Yoffe & McComb), w_low (Yu
6
7 #Packages:
8 import numpy as np
9 import matplotlib.pyplot as plt
10 import csv
11 import math
12
13 #Case Parameters#####
14
15 all_cases = ["C128W", "C128U", "Case2", "HSh", "Decay", "w_low", "w_med", "w_high"]
16
17 #C128W Parameters:
18 k_0_C128W = 14.7
19 eps_0_C128W = 155
20 Sh_C128W = 56.568
21 Sht0_C128W = 12.0
22 t_0_C128W = Sht0_C128W/Sh_C128W
23 t_f_C128W = (15+Sht0_C128W)/Sh_C128W
24 S_0_C128W = Sh_C128W
25 sine_C128W = False
26 w_C128W = None
27
28 #C128U Parameters:
29 k_0_C128U = 6.38
30 eps_0_C128U = 41
31 Sh_C128U = 28.284
32 Sht0_C128U = 8.0
33 t_0_C128U = Sht0_C128U/Sh_C128U
34 t_f_C128U = (10+Sht0_C128U)/Sh_C128U
35 S_0_C128U = Sh_C128U
36 sine_C128U = False
37 w_C128U = None
38
39 #Case2 Parameters:
40 k_0_Case2 = 0.252
41 eps_0_Case2 = 0.234
42 Sh_Case2 = 10*(np.sqrt(2))
43 Sht0_Case2 = 2.0
44 t_0_Case2 = Sht0_Case2/Sh_Case2
45 t_f_Case2 = (2+Sht0_Case2)/Sh_Case2
46 S_0_Case2 = eps_0_Case2/k_0_Case2
47 sine_Case2 = False
48 w_Case2 = None
49
50 #HSh Parameters:
51 n_HSh = 16.7624
52 Re_T_HSh = 1230
53 k_0_HSh = 1
54 nu_HSh = 1E-5
55 q_0_2_HSh = 2*k_0_HSh
56 eps_0_HSh = q_0_2_HSh**2/nu_HSh/Re_T_HSh
57 Sh_HSh = n_HSh/(k_0_HSh/eps_0_HSh)
58 t_0_HSh = 0
59 t_f_HSh = (20/Sh_HSh)
60 S_0_HSh = n_HSh*eps_0_HSh/k_0_HSh
61 sine_HSh = False
62 w_HSh = None
63
64 #Decay Parameters:
65 nu_Decay = 0.0002
66 tau_0_Decay = 1.94
67 Re_lamda_0_Decay = 353.7
68 Re_L_0_Decay = 3828.2
69 U_0_Decay = np.sqrt(Re_L_0_Decay*nu_Decay/tau_0_Decay)
70 k_0_Decay = 3/2*U_0_Decay**2
71 eps_0_Decay = (U_0_Decay**2*np.sqrt(15)/np.sqrt(nu_Decay)/Re_lamda_0_Decay)**2
72 t_0_Decay = 0
73 t_f_Decay = 3.5*tau_0_Decay
74 Sh_Decay = 0
75 S_0_Decay = 1/tau_0_Decay
76 sine_Decay = False
77 w_Decay = None
78
79 #w_low Parameters:
80 S_star_w_low = 3.3
```

```

81 Sh_w_low = 0.0006875 $(Sh_max)
82 k_0_w_low = 1
83 eps_0_w_low = Sh_w_low*k_0_w_low/S_star_w_low
84 S_0_w_low = eps_0_w_low/k_0_w_low
85 t_0_w_low = 0/Sh_w_low
86 t_f_w_low = 50/Sh_w_low
87 sine_w_low = True
88 w_w_low = 0.25*Sh_w_low
89
90 #w_med Parameters:
91 S_star_w_med = 3.3
92 Sh_w_med = 0.0006875 $(Sh_max)
93 k_0_w_med = 1
94 eps_0_w_med = Sh_w_med*k_0_w_med/S_star_w_med
95 S_0_w_med = eps_0_w_med/k_0_w_med
96 t_0_w_med = 0/Sh_w_med
97 t_f_w_med = 50/Sh_w_med
98 sine_w_med = True
99 w_w_med = 0.5*Sh_w_med
100
101 #w_high Parameters:
102 S_star_w_high = 3.3
103 Sh_w_high = 0.0006875 $(Sh_max)
104 k_0_w_high = 1
105 eps_0_w_high = Sh_w_high*k_0_w_high/S_star_w_high
106 S_0_w_high = eps_0_w_high/k_0_w_high
107 t_0_w_high = 0/Sh_w_high
108 t_f_w_high = 50/Sh_w_high
109 sine_w_high = True
110 w_w_high = 1.0*Sh_w_high
111
112 #Model Constants#####
113 n = 1.11
114
115 #Standard k-eps Constants:
116 C_mu_st = 0.09
117 C_eps1_st = 1.44
118 C_eps2_st = 1.92
119
120 #Realizable Constants:
121 A_0_real = 4
122 A_S_real = np.sqrt(6)*np.cos(np.pi/6)
123 C_2_real = (n+1)/n
124
125 #Lumley Constants:
126 A_0_lum = 4
127 A_S_lum = np.sqrt(6)*np.cos(np.pi/6)
128 C_S1_lum = 1/n
129
130 #Standard Reynolds Stress Constants:
131 C_R1_rsms = 1.8
132 C_R2_rsms = 0.6
133 C_eps1_rsms = 1.44
134 C_eps2_rsms = 1.92
135
136 #Realizable Reynolds Stress Constants:
137 C_R1_rsmr = 1.8
138 C_R2_rsmr = 0.6
139 C_2_rsmr = (n+1)/n
140
141 #Lumley Reynolds Stress Constants:
142 C_R1_rsml = 1.8
143 C_R2_rsml = 0.6
144 C_S1_rsml = 1/n
145
146 #Read DNS Data #####
147 for v in all_cases:
148     vars()['k_x_DNS_'+v+'_norm']=[]
149     vars()['k_y_DNS_'+v+'_norm']=[]
150     with open('/home/dtobin/Documents/NASA Research/DNS CSV Data/'+v+'_DNS.csv','r') as csvfile:
151         data = csv.reader(csvfile,delimiter = ',')
152         for row in data:
153             eval('k_x_DNS_'+v+'_norm').append(float(row[0]))
154             eval('k_y_DNS_'+v+'_norm').append(float(row[1]))
155 #Note: HSh is k/k0 vs St, Decay is k/k0 vs t/tau, and Yu & Girimaji is k/ko vs Sh_max*t, but C128W, C128U, Case 2 a
156 k_y_DNS_C128W_norm = [x/k_0_C128W for x in k_y_DNS_C128W_norm]
157 k_y_DNS_C128U_norm = [x/k_0_C128U for x in k_y_DNS_C128U_norm]
158 k_y_DNS_Case2_norm = [x/k_0_Case2 for x in k_y_DNS_Case2_norm]
159
160 #Standard k-eps Solver#####
161 def st_solver (C_mu,C_eps1,C_eps2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):

```

```

162     #Time Domain:
163     dt = (t_f-t_0)/1e3
164
165     #Initialization:
166     t_st_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
167     k_st_dims = [k_0]
168     eps_st_dims = [eps_0]
169     Sh = []
170
171     #Define Lambda Functions:
172     F_k_st = lambda i,y : (C_mu*Sh[i]**2*y**2)/eps_st_dims[i]-eps_st_dims[i]
173     F_eps_st = lambda i,y : ((C_eps1*C_mu*Sh[i]**2*(k_st_dims[i])**2)/y-C_eps2*y)*y/k_st_dims[i]
174
175     #4th Order Runge-Kutta Method:
176     for i in np.arange(0,len(t_st_dims)-1):
177         if sine is False:
178             Sh.append(Sh_max)
179         else:
180             Sh.append(abs(Sh_max*math.sin(w*t_st_dims[i])))
181         for x in ["k","eps"]:
182             lfn = eval("F_"+x+"_st")
183             var = eval(x+"_st_dims")
184             K1 = lfn(i,var[i])
185             K2 = lfn(i,var[i] + 0.5*K1*dt)
186             K3 = lfn(i,var[i] + 0.5*K2*dt)
187             K4 = lfn(i,var[i] + K3*dt)
188             var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
189
190     #Return Data for Plots
191     return (k_st_dims,eps_st_dims,t_st_dims)
192
193 #Realizable k-eps Solver#####
194 def real_solver (A_0,A_S,C_2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
195     #Time Domain:
196     dt = (t_f-t_0)/1e3
197
198     #Initialization:
199     t_real_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
200     k_real_dims = [k_0]
201     eps_real_dims = [eps_0]
202     Sh = []
203     U_star = []
204     eta = []
205     C_1 = []
206     C_mu = []
207
208     #Define Lambda Functions:
209     F_k_real = lambda i,y : (C_mu[i]*Sh[i]**2*y**2)/eps_real_dims[i]-eps_real_dims[i]
210     F_eps_real = lambda i,y : (C_1[i]*abs(Sh[i])*y)-(C_2*y**2/k_real_dims[i])
211
212     #4th Order Runge-Kutta Method:
213     for i in np.arange(0,len(t_real_dims)-1):
214         if sine is False:
215             Sh.append(Sh_max)
216         else:
217             Sh.append(Sh_max*math.sin(w*t_real_dims[i]))
218             U_star.append(abs(Sh[i]))
219             eta.append(abs(Sh[i])*k_real_dims[i]/eps_real_dims[i])
220             C_1.append(max(0.43,eta[i]/(5+eta[i])))
221             C_mu.append(1/(A_0+A_S*U_star[i]*(k_real_dims[i]/eps_real_dims[i])))
222         for x in ["k","eps"]:
223             lfn = eval("F_"+x+"_real")
224             var = eval(x+"_real_dims")
225             K1 = lfn(i,var[i])
226             K2 = lfn(i,var[i] + 0.5*K1*dt)
227             K3 = lfn(i,var[i] + 0.5*K2*dt)
228             K4 = lfn(i,var[i] + K3*dt)
229             var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
230
231     #Return Data for Plots
232     return (k_real_dims,eps_real_dims,t_real_dims)
233
234 #Lumley k-eps Solver#####
235 def lum_solver (A_0,A_S,C_S1,k_0,eps_0,S_0,t_0,t_f,Sh_max,sine,w):
236     #Time Domain:
237     dt = (t_f-t_0)/1e3
238
239     #Initialization:
240     t_lum_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
241     k_lum_dims = [k_0]
242     eps_lum_dims = [eps_0]

```

```

243 S_lum_dims = [S_0]
244 Sh = []
245 U_star = []
246 eta = []
247 C_1 = []
248 C_2 = []
249 C_mu = []
250
251 #Define Lambda Functions:
252 F_k_lum = lambda i,y : (C_mu[i]*Sh[i]**2*y**2)/eps_lum_dims[i]-eps_lum_dims[i]
253 F_eps_lum = lambda i,y : (np.sqrt(2)*C_1[i]*S_lum_dims[i]*y)-((C_2[i]*y**2)/k_lum_dims[i])
254 F_S_lum = lambda i,y : (abs(Sh[i])/np.sqrt(2)-y)*(C_S1*eps_lum_dims[i]/k_lum_dims[i])
255
256 #4th Order Runge-Kutta Method:
257 for i in np.arange(0,len(t_lum_dims)-1):
258     if sine is False:
259         Sh.append(Sh_max)
260     else:
261         Sh.append(Sh_max*math.sin(w*t_lum_dims[i]))
262         U_star.append(abs(Sh[i]))
263         eta.append(abs(Sh[i])*k_lum_dims[i]/eps_lum_dims[i])
264         C_1.append(max(0.43,eta[i]/(5+eta[i])))
265         C_2.append(((n+1)/n)+C_1[i])
266         C_mu.append(1/(A_0+A_S*U_star[i]*(k_lum_dims[i]/eps_lum_dims[i])))
267         for x in ["k","eps","S"]:
268             lfn = eval("F_"+x+"_lum")
269             var = eval(x+"_lum_dims")
270             K1 = lfn(i,var[i])
271             K2 = lfn(i,var[i] + 0.5*K1*dt)
272             K3 = lfn(i,var[i] + 0.5*K2*dt)
273             K4 = lfn(i,var[i] + K3*dt)
274             var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
275
276 #Return Data for Plots
277 return (k_lum_dims,eps_lum_dims,S_lum_dims,t_lum_dims)
278
279 #Standard RSM Solver#####
280 def rsms_solver (C_R1,C_R2,C_eps1,C_eps2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
281     #Time Domain:
282     dt = (t_f-t_0)/1e3
283
284     #Initialization:
285     t_rsms_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
286     k_rsms_dims = [k_0]
287     eps_rsms_dims = [eps_0]
288     uu_rsms_dims = [2*k_0/3]
289     vv_rsms_dims = [2*k_0/3]
290     ww_rsms_dims = [2*k_0/3]
291     uv_rsms_dims = [0]
292     Sh = []
293     b11 = []
294     b22 = []
295
296     #Define Lambda Functions:
297     F_eps_rsms = lambda i,y : -uv_rsms_dims[i]*Sh[i]*y/k_rsms_dims[i]*C_eps1-C_eps2*y**2/k_rsms_dims[i]
298     F_uu_rsms = lambda i,y : -2*(uv_rsms_dims[i]*Sh[i])-(2/3)*(eps_rsms_dims[i])-C_R1*(eps_rsms_dims[i]/k_rsms_dims[i])
299     F_vv_rsms = lambda i,y : (-2/3)*eps_rsms_dims[i]-C_R1*(eps_rsms_dims[i]/k_rsms_dims[i])*(y-(2/3)*k_rsms_dims[i])
300     F_ww_rsms = lambda i,y : (-2/3)*eps_rsms_dims[i]-C_R1*(eps_rsms_dims[i]/k_rsms_dims[i])*(y-(2/3)*k_rsms_dims[i])
301     F_uv_rsms = lambda i,y : -vv_rsms_dims[i]*Sh[i]-C_R1*(eps_rsms_dims[i]/k_rsms_dims[i])*y+C_R2*vv_rsms_dims[i]*S
302
303     #4th Order Runge-Kutta Method:
304     for i in np.arange(0,len(t_rsms_dims)-1):
305         if sine is False:
306             Sh.append(Sh_max)
307         else:
308             Sh.append(Sh_max*math.sin(w*t_rsms_dims[i]))
309         for x in ["eps","uu","vv","ww","uv"]:
310             lfn = eval("F_"+x+"_rsms")
311             var = eval(x+"_rsms_dims")
312             K1 = lfn(i,var[i])
313             K2 = lfn(i,var[i] + 0.5*K1*dt)
314             K3 = lfn(i,var[i] + 0.5*K2*dt)
315             K4 = lfn(i,var[i] + K3*dt)
316             var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
317         k_rsms_dims.append(0.5*(uu_rsms_dims[i+1]+vv_rsms_dims[i+1]+ww_rsms_dims[i+1]))
318         b11.append(uu_rsms_dims[i]/2/k_rsms_dims[i]-(1/3))
319         b22.append(vv_rsms_dims[i]/2/k_rsms_dims[i]-(1/3))
320
321     #Return Data for Plots
322     return (k_rsms_dims,eps_rsms_dims,t_rsms_dims,b11,b22)
323

```

```

324 #Realizable RSM Solver#####
325 def rsmr_solver (C_R1,C_R2,C_2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
326     #Time Domain:
327     dt = (t_f-t_0)/1e3
328
329     #Initialization:
330     t_rsmr_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
331     k_rsmr_dims = [k_0]
332     eps_rsmr_dims = [eps_0]
333     uu_rsmr_dims = [2*k_0/3]
334     vv_rsmr_dims = [2*k_0/3]
335     ww_rsmr_dims = [2*k_0/3]
336     uv_rsmr_dims = [0]
337     Sh = []
338     eta = []
339     C_1 = []
340
341     #Define Lambda Functions:
342     F_eps_rsmr = lambda i,y : (C_1[i]*abs(Sh[i])*y)-(C_2*y**2/k_rsmr_dims[i])
343     F_uu_rsmr = lambda i,y : -2*(uv_rsmr_dims[i]*Sh[i])-(2/3)*(eps_rsmr_dims[i])-(C_R1*(eps_rsmr_dims[i]/k_rsmr_dims
344     F_vv_rsmr = lambda i,y : (-2/3)*eps_rsmr_dims[i]-C_R1*(eps_rsmr_dims[i]/k_rsmr_dims[i])*(y-(2/3)*k_rsmr_dims[i]
345     F_ww_rsmr = lambda i,y : (-2/3)*eps_rsmr_dims[i]-C_R1*(eps_rsmr_dims[i]/k_rsmr_dims[i])*(y-(2/3)*k_rsmr_dims[i]
346     F_uv_rsmr = lambda i,y : -vv_rsmr_dims[i]*Sh[i]-C_R1*(eps_rsmr_dims[i]/k_rsmr_dims[i])*y+C_R2*vv_rsmr_dims[i]*S
347
348     #4th Order Runge-Kutta Method:
349     for i in np.arange(0,len(t_rsmr_dims)-1):
350         if sine is False:
351             Sh.append(Sh_max)
352         else:
353             Sh.append(Sh_max*math.sin(w*t_rsmr_dims[i]))
354         eta.append(abs(Sh[i])*k_rsmr_dims[i]/eps_rsmr_dims[i])
355         C_1.append(max(0.43,eta[i]/(5+eta[i])))
356         for x in ["eps","uu","vv","ww","uv"]:
357             lfn = eval("F_"+x+"_rsmr")
358             var = eval(x+"_rsmr_dims")
359             K1 = lfn(i,var[i])
360             K2 = lfn(i,var[i] + 0.5*K1*dt)
361             K3 = lfn(i,var[i] + 0.5*K2*dt)
362             K4 = lfn(i,var[i] + K3*dt)
363             var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
364         k_rsmr_dims.append(0.5*(uu_rsmr_dims[i+1]+vv_rsmr_dims[i+1]+ww_rsmr_dims[i+1]))
365
366     #Return Data for Plots
367     return (k_rsmr_dims,eps_rsmr_dims,t_rsmr_dims)
368
369 #Lumley RSM Solver#####
370 def rsml_solver (C_R1,C_R2,C_S1,k_0,eps_0,S_0,t_0,t_f,Sh_max,sine,w):
371     #Time Domain:
372     dt = (t_f-t_0)/1e3
373
374     #Initialization:
375     t_rsml_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
376     k_rsml_dims = [k_0]
377     eps_rsml_dims = [eps_0]
378     S_rsml_dims = [S_0]
379     uu_rsml_dims = [2*k_0/3]
380     vv_rsml_dims = [2*k_0/3]
381     ww_rsml_dims = [2*k_0/3]
382     uv_rsml_dims = [0]
383     Sh = []
384     eta = []
385     C_1 = []
386     C_2 = []
387
388     #Define Lambda Functions:
389     F_eps_rsml = lambda i,y : (np.sqrt(2)*C_1[i]*S_rsml_dims[i]*y)-(C_2[i]*y**2/k_rsml_dims[i])
390     F_S_rsml = lambda i,y : (abs(Sh[i])/np.sqrt(2)-y)*(C_S1*eps_rsml_dims[i]/k_rsml_dims[i])
391     F_uu_rsml = lambda i,y : -2*(uv_rsml_dims[i]*Sh[i])-(2/3)*(eps_rsml_dims[i])-(C_R1*(eps_rsml_dims[i]/k_rsml_dims
392     F_vv_rsml = lambda i,y : (-2/3)*eps_rsml_dims[i]-C_R1*(eps_rsml_dims[i]/k_rsml_dims[i])*(y-(2/3)*k_rsml_dims[i]
393     F_ww_rsml = lambda i,y : (-2/3)*eps_rsml_dims[i]-C_R1*(eps_rsml_dims[i]/k_rsml_dims[i])*(y-(2/3)*k_rsml_dims[i]
394     F_uv_rsml = lambda i,y : -vv_rsml_dims[i]*Sh[i]-C_R1*(eps_rsml_dims[i]/k_rsml_dims[i])*y+C_R2*vv_rsml_dims[i]*S
395
396     #4th Order Runge-Kutta Method:
397     for i in np.arange(0,len(t_rsml_dims)-1):
398         if sine is False:
399             Sh.append(Sh_max)
400         else:
401             Sh.append(Sh_max*math.sin(w*t_rsml_dims[i]))
402         eta.append(abs(Sh[i])*k_rsml_dims[i]/eps_rsml_dims[i])
403         C_1.append(max(0.43,eta[i]/(5+eta[i])))
404         C_2.append(((n+1)/n)+C_1[i])

```

```

405     for x in ["eps", "S", "uu", "vv", "ww", "uv"]:
406         lfn = eval("F "+x+"_rsml")
407         var = eval(x+"_rsml_dims")
408         K1 = lfn(i, var[i])
409         K2 = lfn(i, var[i] + 0.5*K1*dt)
410         K3 = lfn(i, var[i] + 0.5*K2*dt)
411         K4 = lfn(i, var[i] + K3*dt)
412         var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
413     k_rsml_dims.append(0.5*(uu_rsml_dims[i+1]+vv_rsml_dims[i+1]+ww_rsml_dims[i+1]))
414
415     #Return Data for Plots
416     return (k_rsml_dims, eps_rsml_dims, S_rsml_dims, t_rsml_dims)
417
418 #Standard k-eps Data#####
419
420 #Call St Function for Each Case and Create Dim. and Norm. k and eps:
421 k_st_dims_dict = {}
422 eps_st_dims_dict = {}
423 t_st_dims_dict = {}
424 k_st_norm_dict = {}
425 eps_st_norm_dict = {}
426 for z in all_cases:
427     st_res = st_solver(C_mu_st, C_eps1_st, C_eps2_st, eval("k_0_"+z), eval("eps_0_"+z), eval("t_0_"+z), eval("t_f_"+z), ev
428     k_st_dims_dict[z] = st_res[0]
429     eps_st_dims_dict[z] = st_res[1]
430     t_st_dims_dict[z] = st_res[2]
431     k_st_norm_dict[z] = [x/eval("k_0_"+z) for x in k_st_dims_dict[z]]
432     eps_st_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_st_dims_dict[z]]
433
434 #Normalize t to Sh(t-t0)
435 t_st_norm_dict = {}
436 for z in ["C128W", "C128U", "Case2"]:
437     t_st_norm_dict[z] = [eval("Sh_"+z)*x - eval("Sht0_"+z) for x in t_st_dims_dict[z]]
438 #Normalize t to Sh(t)
439 for z in ["HSh"]:
440     t_st_norm_dict[z] = [eval("Sh_"+z)*x for x in t_st_dims_dict[z]]
441 #Normalize t to t/tau0
442 for z in ["Decay"]:
443     t_st_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_st_dims_dict[z]]
444 #Normalize t to Sh_max*t
445 for z in ["w_low", "w_med", "w_high"]:
446     t_st_norm_dict[z] = [eval("Sh_"+z)*x for x in t_st_dims_dict[z]]
447
448 #Realizable k-eps Data#####
449
450 #Call Real Function for Each Case and Create Dim. and Norm. k and eps:
451 k_real_dims_dict = {}
452 eps_real_dims_dict = {}
453 t_real_dims_dict = {}
454 k_real_norm_dict = {}
455 eps_real_norm_dict = {}
456 for z in all_cases:
457     real_res = real_solver(A_0_real, A_S_real, C_2_real, eval("k_0_"+z), eval("eps_0_"+z), eval("t_0_"+z), eval("t_f_"+z))
458     k_real_dims_dict[z] = real_res[0]
459     eps_real_dims_dict[z] = real_res[1]
460     t_real_dims_dict[z] = real_res[2]
461     k_real_norm_dict[z] = [x/eval("k_0_"+z) for x in k_real_dims_dict[z]]
462     eps_real_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_real_dims_dict[z]]
463
464 #Normalize t to Sh(t-t0)
465 t_real_norm_dict = {}
466 for z in ["C128W", "C128U", "Case2"]:
467     t_real_norm_dict[z] = [eval("Sh_"+z)*x - eval("Sht0_"+z) for x in t_real_dims_dict[z]]
468 #Normalize t to Sh(t)
469 for z in ["HSh"]:
470     t_real_norm_dict[z] = [eval("Sh_"+z)*x for x in t_real_dims_dict[z]]
471 #Normalize t to t/tau0
472 for z in ["Decay"]:
473     t_real_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_real_dims_dict[z]]
474 #Normalize t to Sh_max*t
475 for z in ["w_low", "w_med", "w_high"]:
476     t_real_norm_dict[z] = [eval("Sh_"+z)*x for x in t_real_dims_dict[z]]
477
478 #Lumley k-eps Data#####
479
480 #Call Lum Function for Each Case and Create Dim. and Norm. k, eps, and S:
481 k_lum_dims_dict = {}
482 eps_lum_dims_dict = {}
483 S_lum_dims_dict = {}
484 t_lum_dims_dict = {}
485 k_lum_norm_dict = {}

```

```

486 eps_lum_norm_dict = {}
487 S_lum_norm_dict = {}
488 for z in all_cases:
489     lum_res = lum_solver(A_0_lum,A_S_lum,C_S1_lum,eval("k_0_"+z),eval("eps_0_"+z),eval("S_0_"+z),eval("t_0_"+z),eva
490     k_lum_dims_dict[z] = lum_res[0]
491     eps_lum_dims_dict[z] = lum_res[1]
492     S_lum_dims_dict[z] = lum_res[2]
493     t_lum_dims_dict[z] = lum_res[3]
494     k_lum_norm_dict[z] = [x/eval("k_0_"+z) for x in k_lum_dims_dict[z]]
495     eps_lum_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_lum_dims_dict[z]]
496     S_lum_norm_dict[z] = [x/eval("S_0_"+z) for x in S_lum_dims_dict[z]]
497
498 #Normalize t to Sh(t-t0)
499 t_lum_norm_dict = {}
500 for z in ["C128W","C128U","Case2"]:
501     t_lum_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_lum_dims_dict[z]]
502 #Normalize t to Sh(t)
503 for z in ["HSh"]:
504     t_lum_norm_dict[z] = [eval("Sh_"+z)*x for x in t_lum_dims_dict[z]]
505 #Normalize t to t/tau0
506 for z in ["Decay"]:
507     t_lum_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_lum_dims_dict[z]]
508 #Normalize t to Sh_max*t
509 for z in ["w_low","w_med","w_high"]:
510     t_lum_norm_dict[z] = [eval("Sh_"+z)*x for x in t_lum_dims_dict[z]]
511
512 #Standard RSM Data#####
513
514 #Call RSMS Function for Each Case and Create Dim. and Norm. k and eps:
515 k_rsms_dims_dict = {}
516 eps_rsms_dims_dict = {}
517 t_rsms_dims_dict = {}
518 k_rsms_norm_dict = {}
519 eps_rsms_norm_dict = {}
520 b11_rsms_dims_dict = {}
521 b22_rsms_dims_dict = {}
522 for z in all_cases:
523     rsms_res = rsms_solver(C_R1_rsms,C_R2_rsms,C_eps1_rsms,C_eps2_rsms,eval("k_0_"+z),eval("eps_0_"+z),eval("t_0_"+
524     k_rsms_dims_dict[z] = rsms_res[0]
525     eps_rsms_dims_dict[z] = rsms_res[1]
526     t_rsms_dims_dict[z] = rsms_res[2]
527     b11_rsms_dims_dict[z] = rsms_res[3]
528     b22_rsms_dims_dict[z] = rsms_res[4]
529     k_rsms_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsms_dims_dict[z]]
530     eps_rsms_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsms_dims_dict[z]]
531
532 #Normalize t to Sh(t-t0)
533 t_rsms_norm_dict = {}
534 for z in ["C128W","C128U","Case2"]:
535     t_rsms_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsms_dims_dict[z]]
536 #Normalize t to Sh(t)
537 for z in ["HSh"]:
538     t_rsms_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsms_dims_dict[z]]
539 #Normalize t to t/tau0
540 for z in ["Decay"]:
541     t_rsms_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsms_dims_dict[z]]
542 #Normalize t to Sh_max*t
543 for z in ["w_low","w_med","w_high"]:
544     t_rsms_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsms_dims_dict[z]]
545
546 #Realizable RSM Data#####
547
548 #Call RSMR Function for Each Case and Create Dim. and Norm. k and eps:
549 k_rsmr_dims_dict = {}
550 eps_rsmr_dims_dict = {}
551 t_rsmr_dims_dict = {}
552 k_rsmr_norm_dict = {}
553 eps_rsmr_norm_dict = {}
554 for z in all_cases:
555     rsmr_res = rsmr_solver(C_R1_rsmr,C_R2_rsmr,C_2_rsmr,eval("k_0_"+z),eval("eps_0_"+z),eval("t_0_"+z),eval("t_f_"+
556     k_rsmr_dims_dict[z] = rsmr_res[0]
557     eps_rsmr_dims_dict[z] = rsmr_res[1]
558     t_rsmr_dims_dict[z] = rsmr_res[2]
559     k_rsmr_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsmr_dims_dict[z]]
560     eps_rsmr_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsmr_dims_dict[z]]
561
562 #Normalize t to Sh(t-t0)
563 t_rsmr_norm_dict = {}
564 for z in ["C128W","C128U","Case2"]:
565     t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsmr_dims_dict[z]]
566 #Normalize t to Sh(t)

```

```

567 for z in ["HSh"]:
568     t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsmr_dims_dict[z]]
569 #Normalize t to t/tau0
570 for z in ["Decay"]:
571     t_rsmr_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsmr_dims_dict[z]]
572 #Normalize t to Sh_max*t
573 for z in ["w_low","w_med","w_high"]:
574     t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsmr_dims_dict[z]]
575
576 #Lumley RSM Data#####
577
578 #Call RSML Function for Each Case and Create Dim. and Norm. k, eps, and S:
579 k_rsm_l_dims_dict = {}
580 eps_rsm_l_dims_dict = {}
581 S_rsm_l_dims_dict = {}
582 t_rsm_l_dims_dict = {}
583 k_rsm_l_norm_dict = {}
584 eps_rsm_l_norm_dict = {}
585 S_rsm_l_norm_dict = {}
586 for z in all_cases:
587     rsm_l_res = rsm_l_solver(C_R1_rsm_l,C_R2_rsm_l,C_S1_rsm_l,eval("k_0_"+z),eval("eps_0_"+z),eval("S_0_"+z),eval("t_0_"
588     k_rsm_l_dims_dict[z] = rsm_l_res[0]
589     eps_rsm_l_dims_dict[z] = rsm_l_res[1]
590     S_rsm_l_dims_dict[z] = rsm_l_res[2]
591     t_rsm_l_dims_dict[z] = rsm_l_res[3]
592     k_rsm_l_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsm_l_dims_dict[z]]
593     eps_rsm_l_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsm_l_dims_dict[z]]
594     S_rsm_l_norm_dict[z] = [x/eval("S_0_"+z) for x in S_rsm_l_dims_dict[z]]
595
596 #Normalize t to Sh(t-t0)
597 t_rsm_l_norm_dict = {}
598 for z in ["C128W","C128U","Case2"]:
599     t_rsm_l_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsm_l_dims_dict[z]]
600 #Normalize t to Sh(t)
601 for z in ["HSh"]:
602     t_rsm_l_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsm_l_dims_dict[z]]
603 #Normalize t to t/tau0
604 for z in ["Decay"]:
605     t_rsm_l_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsm_l_dims_dict[z]]
606 #Normalize t to Sh_max*t
607 for z in ["w_low","w_med","w_high"]:
608     t_rsm_l_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsm_l_dims_dict[z]]
609
610 #CREATE PLOTS:#####
611 plot_L = 11
612 plot_W = 6
613 plot_DPI = 200
614 mpl.rcParams.update({'font.size': 12})
615
616 #####C128W NORMALIZED PLOTS:#####
617 mpl.figure(1)
618 mpl.suptitle('C128W - Rogers $(S_s = '+str(round(Sh_C128W,2))+')$')
619
620 #mpl.subplot(1,2,1)
621 #DNS:
622 mpl.plot (k_x_DNS_C128W_norm,k_y_DNS_C128W_norm,'s',color='white',markeredgecolor='black',label='DNS')
623 #Standard k-eps:
624 mpl.plot (t_st_norm_dict['C128W'],k_st_norm_dict['C128W'],'--',color = 'blue',label='Standard $k-\backslash\epsilon$')
625 #Realizable k-eps:
626 mpl.plot (t_real_norm_dict['C128W'],k_real_norm_dict['C128W'],'--',color = 'red',label='Realizable $k-\backslash\epsilon$')
627 #Lumley k-eps:
628 mpl.plot (t_lum_norm_dict['C128W'],k_lum_norm_dict['C128W'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
629 #RSMs:
630 mpl.plot (t_rsms_norm_dict['C128W'],k_rsms_norm_dict['C128W'],color = 'blue',label='Standard RSM')
631 #RSMR:
632 mpl.plot (t_rsmr_norm_dict['C128W'],k_rsmr_norm_dict['C128W'],color = 'red',label='Realizable RSM')
633 #RSML:
634 mpl.plot (t_rsm_l_norm_dict['C128W'],k_rsm_l_norm_dict['C128W'],color = 'green',label='Lumley RSM')
635
636 #Formatting(1,1):
637 mpl.legend(loc='upper left',ncol=1)
638 mpl.xlabel("$S_s*(t-t_0)$")
639 mpl.ylabel("$k/k_0$")
640 mpl.xlim (0,15)
641 mpl.ylim (0,200/k_0_C128W)
642 mpl.xticks (np.linspace(0,15,4))
643
644 #Uncomment to plot S as a function of time:
645 # ax = mpl.subplot(1,2,2)
646 # #Lumley k-eps:
647 # mpl.plot (t_lum_norm_dict['C128W'],S_lum_norm_dict['C128W'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')

```

```

648 # #Target Value:
649 # mpl.plot(t_lum_norm_dict['C128W'],[Sh_C128W/S_0_C128W for x in t_lum_norm_dict['C128W']], '--',color='black',label=
650
651 # #Formating(1,2):
652 # mpl.legend(loc='upper right',ncol=1)
653 # mpl.xlabel("$S_s*(t-t_0)$")
654 # mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
655 # mpl.xlim (0,15)
656 # mpl.xticks (np.linspace(0,15,4))
657 # ax.yaxis.tick_right()
658 # ax.yaxis.set_label_position("right")
659
660 fig = mpl.gcf()
661 fig.set_size_inches(plot_L, plot_W)
662 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/C128W.png',dpi = plot_DPI)
663
664 #####C128U NORMALIZED PLOTS:#####
665 mpl.figure(2)
666 mpl.suptitle('C128U - Rogers $(S_s = '+str(round(Sh_C128U,2))+')$')
667
668 #mpl.subplot(1,2,1)
669 #DNS:
670 mpl.plot (k_x_DNS_C128U_norm,k_y_DNS_C128U_norm,'s',color='white',markeredgecolor='black',label='DNS')
671 #Standard k-eps:
672 mpl.plot (t_st_norm_dict['C128U'],k_st_norm_dict['C128U'],'--',color = 'blue',label='Standard $k-\\epsilon$')
673 #Realizable k-eps:
674 mpl.plot (t_real_norm_dict['C128U'],k_real_norm_dict['C128U'],'--',color = 'red',label='Realizable $k-\\epsilon$')
675 #Lumley k-eps:
676 mpl.plot (t_lum_norm_dict['C128U'],k_lum_norm_dict['C128U'],'--',color = 'green',label='Lumley $k-\\epsilon$')
677 #RSMs:
678 mpl.plot (t_rsms_norm_dict['C128U'],k_rsms_norm_dict['C128U'],color = 'blue',label='Standard RSM')
679 #RSMR:
680 mpl.plot (t_rsmr_norm_dict['C128U'],k_rsmr_norm_dict['C128U'],color = 'red',label='Realizable RSM')
681 #RSML:
682 mpl.plot (t_rsm_l_norm_dict['C128U'],k_rsm_l_norm_dict['C128U'],color = 'green',label='Lumley RSM')
683
684 #Formating(2,1):
685 mpl.legend(loc='upper left',ncol=1)
686 mpl.xlabel("$S_s*(t-t_0)$")
687 mpl.ylabel("$k/k_0$")
688 mpl.xlim (0,10)
689 mpl.ylim (0,30/k_0_C128U)
690 mpl.xticks (np.linspace(0,10,3))
691
692 #Uncomment to plot S as a function of time:
693 # ax = mpl.subplot(1,2,2)
694 # #Lumley k-eps:
695 # mpl.plot (t_lum_norm_dict['C128U'],S_lum_norm_dict['C128U'],'--',color = 'green',label='Lumley $k-\\epsilon$')
696 # #Target Value:
697 # mpl.plot(t_lum_norm_dict['C128U'],[Sh_C128U/S_0_C128U for x in t_lum_norm_dict['C128U']], '--',color='black',label=
698
699 # #Formating(2,2):
700 # mpl.legend(loc='upper right',ncol=1)
701 # mpl.xlabel("$S_s*(t-t_0)$")
702 # mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
703 # mpl.xlim (0,10)
704 # mpl.xticks (np.linspace(0,10,3))
705 # ax.yaxis.tick_right()
706 # ax.yaxis.set_label_position("right")
707
708 fig = mpl.gcf()
709 fig.set_size_inches(plot_L, plot_W)
710 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/C128U.png',dpi = plot_DPI)
711
712 #####Case2 NORMALIZED PLOTS:#####
713 mpl.figure(3)
714 mpl.suptitle('Case2 - Matsumoto $(S_s = '+str(round(Sh_Case2,2))+')$')
715
716 #mpl.subplot(1,2,1)
717 #DNS:
718 mpl.plot (k_x_DNS_Case2_norm,k_y_DNS_Case2_norm,'s',color='white',markeredgecolor='black',label='DNS')
719 #Standard k-eps:
720 mpl.plot (t_st_norm_dict['Case2'],k_st_norm_dict['Case2'],'--',color = 'blue',label='Standard $k-\\epsilon$')
721 #Realizable k-eps:
722 mpl.plot (t_real_norm_dict['Case2'],k_real_norm_dict['Case2'],'--',color = 'red',label='Realizable $k-\\epsilon$')
723 #Lumley k-eps:
724 mpl.plot (t_lum_norm_dict['Case2'],k_lum_norm_dict['Case2'],'--',color = 'green',label='Lumley $k-\\epsilon$')
725 #RSMs:
726 mpl.plot (t_rsms_norm_dict['Case2'],k_rsms_norm_dict['Case2'],color = 'blue',label='Standard RSM')
727 #RSMR:
728 mpl.plot (t_rsmr_norm_dict['Case2'],k_rsmr_norm_dict['Case2'],color = 'red',label='Realizable RSM')

```

```

729 #RSM:
730 mpl.plot (t_rsm_norm_dict['Case2'],k_rsm_norm_dict['Case2'],color = 'green',label='Lumley RSM')
731
732 #Formating(3,1):
733 mpl.legend(loc='upper left',ncol=1)
734 mpl.xlabel("$S_s*(t-t_0)$")
735 mpl.ylabel("$k/k_0$")
736 mpl.xlim (0,2)
737 mpl.ylim (0,0.7/k_0_Case2)
738 mpl.xticks (np.linspace(0,2,3))
739
740 #Uncomment to plot S as a function of time:
741 # ax = mpl.subplot(1,2,2)
742 # #Lumley k-eps:
743 # mpl.plot (t_lum_norm_dict['Case2'],S_lum_norm_dict['Case2'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
744 # #Target Value:
745 # mpl.plot(t_lum_norm_dict['Case2'],[Sh_Case2/S_0_Case2 for x in t_lum_norm_dict['Case2']], '-',color='black',label=
746
747 # #Formating(3,2):
748 # mpl.legend(loc='upper right',ncol=1)
749 # mpl.xlabel("$S_s*(t-t_0)$")
750 # mpl.ylabel("$\mathcal{S}/\mathcal{S}_0$")
751 # mpl.xlim (0,2)
752 # mpl.xticks (np.linspace(0,2,3))
753 # ax.yaxis.tick_right()
754 # ax.yaxis.set_label_position("right")
755
756 fig = mpl.gcf()
757 fig.set_size_inches(plot_L, plot_W)
758 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/Case2.png',dpi = plot_DPI)
759
760 #####HSh NORMALIZED PLOTS:#####
761 mpl.figure(4)
762 mpl.suptitle('HSh - Lee $(S_s = '+str(round(Sh_HSh,2))+')$')
763
764 #mpl.subplot(1,2,1)
765 #DNS:
766 mpl.plot (k_x_DNS_HSh_norm,k_y_DNS_HSh_norm,'s',color='white',markeredgecolor='black',label='DNS')
767 #Standard k-eps:
768 mpl.plot (t_st_norm_dict['HSh'],k_st_norm_dict['HSh'],'--',color = 'blue',label='Standard $k-\backslash\epsilon$')
769 #Realizable k-eps:
770 mpl.plot (t_real_norm_dict['HSh'],k_real_norm_dict['HSh'],'--',color = 'red',label='Realizable $k-\backslash\epsilon$')
771 #Lumley k-eps:
772 mpl.plot (t_lum_norm_dict['HSh'],k_lum_norm_dict['HSh'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
773 #RSMs:
774 mpl.plot (t_rsms_norm_dict['HSh'],k_rsms_norm_dict['HSh'],color = 'blue',label='Standard RSM')
775 #RSMR:
776 mpl.plot (t_rsmr_norm_dict['HSh'],k_rsmr_norm_dict['HSh'],color = 'red',label='Realizable RSM')
777 #RSM:
778 mpl.plot (t_rsm_norm_dict['HSh'],k_rsm_norm_dict['HSh'],color = 'green',label='Lumley RSM')
779
780 #Formating(4,1):
781 mpl.legend(loc='upper left',ncol=1)
782 mpl.xlabel("$S_s*t$")
783 mpl.ylabel("$k/k_0$")
784 mpl.xlim (0,17)
785 mpl.ylim (0,20)
786 mpl.xticks ([0,5,10,15])
787 mpl.yticks(np.linspace(0,20,5))
788
789 #Uncomment to plot S as a function of time:
790 # ax = mpl.subplot(1,2,2)
791 # #Lumley k-eps:
792 # mpl.plot (t_lum_norm_dict['HSh'],S_lum_norm_dict['HSh'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
793 # #Target Value:
794 # mpl.plot(t_lum_norm_dict['HSh'],[Sh_HSh/S_0_HSh for x in t_lum_norm_dict['HSh']], '-',color='black',label='Target
795
796 # #Formating(4,2):
797 # mpl.legend(loc='upper right',ncol=1)
798 # mpl.xlabel("$S_s*t$")
799 # mpl.ylabel("$\mathcal{S}/\mathcal{S}_0$")
800 # mpl.xlim (0,17)
801 # mpl.xticks ([0,5,10,15])
802 # ax.yaxis.tick_right()
803 # ax.yaxis.set_label_position("right")
804
805 fig = mpl.gcf()
806 fig.set_size_inches(plot_L, plot_W)
807 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/HSh.png',dpi = plot_DPI)
808
809 #####Decay NORMALIZED PLOTS:#####

```

```

810 mpl.figure(5)
811 mpl.suptitle('Isotropic Decay - Yoffe & McComb $(S_s = '+str(round(Sh_Decay,2))+')$')
812
813 #mpl.subplot(1,2,1)
814 #DNS:
815 mpl.plot (k_x_DNS_Decay_norm,k_y_DNS_Decay_norm,'s',color='white',markeredgecolor='black',label='DNS')
816 #Standard k-eps:
817 mpl.plot (t_st_norm_dict['Decay'],k_st_norm_dict['Decay'],'--',color = 'blue',label='Standard $k-\backslash\epsilon$')
818 #Realizable k-eps:
819 mpl.plot (t_real_norm_dict['Decay'],k_real_norm_dict['Decay'],'--',color = 'red',label='Realizable $k-\backslash\epsilon$')
820 #Lumley k-eps:
821 mpl.plot (t_lum_norm_dict['Decay'],k_lum_norm_dict['Decay'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
822 #RSMS:
823 mpl.plot (t_rsms_norm_dict['Decay'],k_rsms_norm_dict['Decay'],color = 'blue',label='Standard RSM')
824 #RSMR:
825 mpl.plot (t_rsmr_norm_dict['Decay'],k_rsmr_norm_dict['Decay'],color = 'red',label='Realizable RSM')
826 #RSMML:
827 mpl.plot (t_rsmml_norm_dict['Decay'],k_rsmml_norm_dict['Decay'],color = 'green',label='Lumley RSM')
828
829 #Formating(5,1):
830 mpl.legend(loc='upper right',ncol=1)
831 mpl.xlabel("$t/\backslash\tau_0$")
832 mpl.ylabel("$k/k_0$")
833 mpl.xlim (0,3.5)
834 mpl.ylim (0.2,1.1)
835 mpl.xticks (np.linspace(0,3.5,8))
836 mpl.yticks(np.linspace(0.2,1.1,10))
837
838 #Uncomment to plot S as a function of time:
839 # ax = mpl.subplot(1,2,2)
840 # #Lumley k-eps:
841 # mpl.plot (t_lum_norm_dict['Decay'],S_lum_norm_dict['Decay'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
842 # #Target Value:
843 # mpl.plot(t_lum_norm_dict['Decay'],[Sh_Decay/S_0_Decay for x in t_lum_norm_dict['Decay']], '-',color='black',label=
844
845 # #Formating(5,2):
846 # mpl.legend(loc='upper right',ncol=1)
847 # mpl.xlabel("$t/\backslash\tau_0$")
848 # mpl.ylabel("$\backslash\mathcal{S}/\backslash\mathcal{S}_0$")
849 # mpl.xlim (0,3.5)
850 # mpl.xticks (np.linspace(0,3.5,8))
851 # ax.yaxis.tick_right()
852 # ax.yaxis.set_label_position("right")
853
854 fig = mpl.gcf()
855 fig.set_size_inches(plot_L, plot_W)
856 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/Decay.png',dpi = plot_DPI)
857
858 #####w_low NORMALIZED PLOTS:#####
859 mpl.figure(6)
860 mpl.suptitle('$\backslash\omega/S_{\max} = 0.25$ - Yu & Girimaji $(S_s = S_{\max}\backslash\sin\{\backslash\omega t\})$')
861
862 #mpl.subplot(1,2,1)
863 #DNS:
864 mpl.semilogy (k_x_DNS_w_low_norm,k_y_DNS_w_low_norm,'s',color='white',markeredgecolor='black',label='DNS')
865 #Standard k-eps:
866 mpl.semilogy (t_st_norm_dict['w_low'],k_st_norm_dict['w_low'],'--',color = 'blue',label='Standard $k-\backslash\epsilon$')
867 #Realizable k-eps:
868 mpl.semilogy (t_real_norm_dict['w_low'],k_real_norm_dict['w_low'],'--',color = 'red',label='Realizable $k-\backslash\epsilon$')
869 #Lumley k-eps:
870 mpl.semilogy (t_lum_norm_dict['w_low'],k_lum_norm_dict['w_low'],'--',color = 'green',label='Lumley $k-\backslash\epsilon$')
871 #RSMS:
872 mpl.semilogy (t_rsms_norm_dict['w_low'],k_rsms_norm_dict['w_low'],color = 'blue',label='Standard RSM')
873 #RSMR:
874 mpl.semilogy (t_rsmr_norm_dict['w_low'],k_rsmr_norm_dict['w_low'],color = 'red',label='Realizable RSM')
875 #RSMML:
876 mpl.semilogy (t_rsmml_norm_dict['w_low'],k_rsmml_norm_dict['w_low'],color = 'green',label='Lumley RSM')
877
878 #Formating(6,1):
879 mpl.legend(loc='upper left',ncol=1)
880 mpl.xlabel("$S_{\max}*t$")
881 mpl.ylabel("$k/k_0$")
882 mpl.xlim (0,50)
883 #mpl.ylim (0,20)
884 mpl.xticks (np.linspace(0,50,6))
885 #mpl.yticks(np.linspace(0,20,3))
886
887 #Uncomment to plot S as a function of time:
888 # ax = mpl.subplot(1,2,2)
889 # #Lumley k-eps:
890 # mpl.plot (t_lum_norm_dict['w_low'],[x/Sh_w_low for x in S_lum_dims_dict['w_low']], '--',color = 'green',label=' $

```

```

891 # #Sh:
892 # mpl.plot(t_lum_norm_dict['w_low'],[(math.sin(w_w_low*x)) for x in t_lum_dims_dict['w_low']], '--',color='black',lat
893
894 # #Formating(6,2):
895 # mpl.legend(loc='upper right',ncol=1)
896 # mpl.xlabel("$S_{max}*t$")
897 # mpl.xlim (0,50)
898 # mpl.xticks (np.linspace(0,50,6))
899 # mpl.ylim (-3,3)
900 # ax.yaxis.tick_right()
901 # ax.yaxis.set_label_position("right")
902
903 fig = mpl.gcf()
904 fig.set_size_inches(plot_L, plot_W)
905 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_low.png',dpi = plot_DPI)
906
907 #####w_med NORMALIZED PLOTS:#####
908 mpl.figure(7)
909 mpl.suptitle('$\\omega/S_{max} = 0.5$ - Yu & Girimaji ($S_s = S_{max}\\sin{\\omega t}$)')
910
911 #mpl.subplot(1,2,1)
912 #DNS:
913 mpl.semilogy (k_x_DNS_w_med_norm,k_y_DNS_w_med_norm,'s',color='white',markeredgecolor='black',label='DNS')
914 #Standard k-eps:
915 mpl.semilogy (t_st_norm_dict['w_med'],k_st_norm_dict['w_med'],'--',color = 'blue',label='Standard $k-\\epsilon$')
916 #Realizable k-eps:
917 mpl.semilogy (t_real_norm_dict['w_med'],k_real_norm_dict['w_med'],'--',color = 'red',label='Realizable $k-\\epsilon$')
918 #Lumley k-eps:
919 mpl.semilogy (t_lum_norm_dict['w_med'],k_lum_norm_dict['w_med'],'--',color = 'green',label='Lumley $k-\\epsilon$')
920 #RSMS:
921 mpl.semilogy (t_rsms_norm_dict['w_med'],k_rsms_norm_dict['w_med'],color = 'blue',label='Standard RSM')
922 #RSMR:
923 mpl.semilogy (t_rsmr_norm_dict['w_med'],k_rsmr_norm_dict['w_med'],color = 'red',label='Realizable RSM')
924 #RSMML:
925 mpl.semilogy (t_rsmml_norm_dict['w_med'],k_rsmml_norm_dict['w_med'],color = 'green',label='Lumley RSM')
926
927 #Formating(7,1):
928 mpl.legend(loc='upper left',ncol=1)
929 mpl.xlabel("$S_{max}*t$")
930 mpl.ylabel("$k/k_0$")
931 mpl.xlim (0,50)
932 #mpl.ylim (0,1)
933 mpl.xticks (np.linspace(0,50,6))
934 #mpl.yticks(np.linspace(0,1,3))
935
936 #Uncomment to plot S as a function of time:
937 # ax = mpl.subplot(1,2,2)
938 # #Lumley k-eps:
939 # mpl.plot (t_lum_norm_dict['w_med'],[x/Sh_w_med for x in S_lum_dims_dict['w_med']], '--',color = 'green',label=' $)
940 # #Sh:
941 # mpl.plot(t_lum_norm_dict['w_med'],[(math.sin(w_w_med*x)) for x in t_lum_dims_dict['w_low']], '--',color='black',lat
942
943 # #Formating(7,2):
944 # mpl.legend(loc='upper right',ncol=1)
945 # mpl.xlabel("$S_{max}*t$")
946 # mpl.xlim (0,50)
947 # mpl.xticks (np.linspace(0,50,6))
948 # mpl.ylim (-3,3)
949 # ax.yaxis.tick_right()
950 # ax.yaxis.set_label_position("right")
951
952 fig = mpl.gcf()
953 fig.set_size_inches(plot_L, plot_W)
954 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_med.png',dpi = plot_DPI)
955
956 #####w_high NORMALIZED PLOTS:#####
957 mpl.figure(8)
958 mpl.suptitle('$\\omega/S_{max} = 1.0$ - Yu & Girimaji ($S_s = S_{max}\\sin{\\omega t}$)')
959
960 #mpl.subplot(1,2,1)
961 #DNS:
962 mpl.semilogy (k_x_DNS_w_high_norm,k_y_DNS_w_high_norm,'s',color='white',markeredgecolor='black',label='DNS')
963 #Standard k-eps:
964 mpl.semilogy (t_st_norm_dict['w_high'],k_st_norm_dict['w_high'],'--',color = 'blue',label='Standard $k-\\epsilon$')
965 #Realizable k-eps:
966 mpl.semilogy (t_real_norm_dict['w_high'],k_real_norm_dict['w_high'],'--',color = 'red',label='Realizable $k-\\epsilon$')
967 #Lumley k-eps:
968 mpl.semilogy (t_lum_norm_dict['w_high'],k_lum_norm_dict['w_high'],'--',color = 'green',label='Lumley $k-\\epsilon$')
969 #RSMS:
970 mpl.semilogy (t_rsms_norm_dict['w_high'],k_rsms_norm_dict['w_high'],color = 'blue',label='Standard RSM')
971 #RSMR:

```

```

972 mpl.semilogy (t_rsmr_norm_dict['w_high'],k_rsmr_norm_dict['w_high'],color = 'red',label='Realizable RSM')
973 #RSM:
974 mpl.semilogy (t_rsml_norm_dict['w_high'],k_rsml_norm_dict['w_high'],color = 'green',label='Lumley RSM')
975
976 #Formating(8,1):
977 mpl.legend(loc='upper left',ncol=1)
978 mpl.xlabel("$S_{max}*t$")
979 mpl.ylabel("$k/k_0$")
980 mpl.xlim (0,50)
981 #mpl.ylim (0,1)
982 mpl.xticks (np.linspace(0,50,6))
983 #mpl.yticks(np.linspace(0,1,3))
984
985 #Uncomment to plot S as a function of time:
986 # ax = mpl.subplot(1,2,2)
987 # #Lumley k-eps:
988 # mpl.plot (t_lum_norm_dict['w_high'],[x/Sh_w_high for x in S_lum_dims_dict['w_high']], '--',color = 'green',label='
989 # #Sh:
990 # mpl.plot(t_lum_norm_dict['w_high'],[(math.sin(w_w_high*x)) for x in t_lum_dims_dict['w_low']], '- ',color='black',l
991
992 # #Formating(8,2):
993 # mpl.legend(loc='upper right',ncol=1)
994 # mpl.xlabel("$S_{max}*t$")
995 # mpl.xlim (0,50)
996 # mpl.xticks (np.linspace(0,50,6))
997 # mpl.ylim (-3,3)
998 # ax.yaxis.tick_right()
999 # ax.yaxis.set_label_position("right")
1000
1001 fig = mpl.gcf()
1002 fig.set_size_inches(plot_L, plot_W)
1003 mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_high.png',dpi = plot_DPI)
1004
1005 #mpl.close("all")
1006 #####b11/b22 Test Plots#####
1007 mpl.figure(9)
1008 mpl.title("$b_{11}$")
1009 mpl.plot (t_rsms_norm_dict['w_med'][0:999],b11_rsms_dims_dict['w_med'],label = "$\\omega / S_{max} = 0.5$")
1010 mpl.plot (t_rsms_norm_dict['w_high'][0:999],b11_rsms_dims_dict['w_high'],label = "$\\omega / S_{max} = 1.0$")
1011 mpl.legend(loc='upper right',ncol=1)
1012 mpl.xlabel("$S_{max}*t$")
1013 mpl.ylabel("$b_{11}$")
1014 mpl.xlim (0,50)
1015 mpl.xticks (np.linspace(0,50,6))
1016
1017 mpl.figure(10)
1018 mpl.title("$b_{22}$")
1019 mpl.plot (t_rsms_norm_dict['w_med'][0:999],b22_rsms_dims_dict['w_med'],label = "$\\omega / S_{max} = 0.5$")
1020 mpl.plot (t_rsms_norm_dict['w_high'][0:999],b22_rsms_dims_dict['w_high'],label = "$\\omega / S_{max} = 1.0$")
1021 mpl.legend(loc='upper right',ncol=1)
1022 mpl.xlabel("$S_{max}*t$")
1023 mpl.ylabel("$b_{22}$")
1024 mpl.xlim (0,50)
1025 mpl.xticks (np.linspace(0,50,6))
1026
1027 #####Show Plots:####
1028 mpl.show(block=False)

```