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 model[2] 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

List of Figures

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 1991[2]. 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 Durbin[1] (p. 122), the general transport equation for k is:

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

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

$$\frac{\partial k}{\partial t}=\mathcal{P}-\varepsilon \tag{2}$$

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

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

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

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

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

Substituting into (2) gives:

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

From Durbin[1] (p. 122), the general transport equation for ε is:

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

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

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

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

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

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:

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

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:

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

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

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

$$\Omega_{ij} = \overline{\Omega}_{ij} - \varepsilon_{ijk}\omega_k \tag{16}$$

$$A_0 = 4.04 \tag{17}$$

$$A_S = \sqrt{6}\cos\phi \tag{18}$$

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

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

For parallel shear flow:

$$W = 0 \tag{21}$$

$$\phi = \frac{\pi}{6} \tag{22}$$

$$A_S = \frac{3\sqrt{2}}{2} \tag{23}$$

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

The proposed transport equation for ε is:

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

The equation is reduced as follows:

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

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

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

The new constants C_1 and C_2 are proposed to be:

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

$$\eta = |S_s|\frac{k}{\varepsilon} \tag{30}$$

$$C_2 = \frac{n+1}{n} \tag{31}$$

$$n = 1.11 \tag{32}$$

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 Lumley[2] (1992):

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

This equation can be reduced as follows:

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

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

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

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

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:

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

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:

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

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

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

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:

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

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

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

$$\frac{1}{t_0} = C_{S1}\frac{nk_0}{t_0 k_0} \qquad \left(\text{because} \quad \varepsilon_0 = \frac{nk_0}{t_0}\right) \tag{44}$$

$$\boxed{C_{S1} = \frac{1}{n}} \tag{45}$$

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

Standard Reynolds Stress Model ($\mathbf{RSM_S}$)

The general Reynolds stress model transport equation[4] (assuming no diffusion and no production by body forces) is:

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

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

$$\varepsilon_{ij} = \frac{2}{3} \varepsilon \partial_{ij} \tag{48}$$

$$\Phi_{ij} = \Phi_{ij_1} + \Phi_{ij_2} \tag{49}$$

$$\Phi_{ij_1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_i u_j} - \frac{2}{3} \partial_{ij} k \right) \tag{50}$$

$$\Phi_{ij_2} = -C_{R2} \left(P_{ij} - \frac{2}{3} \partial_{ij} P_k \right) \tag{51}$$

$$P_k = -\overline{u_i u_k} \frac{\partial u_i}{\partial x_k} \tag{52}$$

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

$$\boxed{k = \frac{1}{2} \left[\overline{u_1 u_1} + \overline{u_2 u_2} + \overline{u_3 u_3} \right]} \tag{53}$$

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

$\underline{i = 1 \quad j = 1:}$

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

$$\varepsilon_{11} = \frac{2}{3} \varepsilon \tag{55}$$

$$\Phi_{11_1} = -C_{R1} \frac{\varepsilon}{k} \left(\overline{u_1 u_1} - \frac{2}{3} k \right) \tag{56}$$

$$\Phi_{11_2} = C_{R2} \frac{4}{3} \overline{u_1 u_2} S_s \tag{57}$$

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

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

$\underline{i=2 \quad j=2:}$

$$P_{22} = 0 \tag{60}$$

$$\varepsilon_{22} = \frac{2}{3}\varepsilon \tag{61}$$

$$\Phi_{22_1} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_2u_2} - \frac{2}{3}k\right) \tag{62}$$

$$\Phi_{22_2} = -C_{R2}\frac{2}{3}\overline{u_1u_2}S_s \tag{63}$$

$$\Phi_{22} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_2u_2} - \frac{2}{3}k\right) - C_{R2}\frac{2}{3}\overline{u_1u_2}S_s \tag{64}$$

$$\boxed{\frac{\partial\overline{u_2u_2}}{\partial t} = -\frac{2}{3}\varepsilon - C_{R1}\frac{\varepsilon}{k}\left(\overline{u_2u_2} - \frac{2}{3}k\right) - C_{R2}\frac{2}{3}\overline{u_1u_2}S_s} \tag{65}$$

$\underline{i=3 \quad j=3:}$

$$P_{33} = 0 \tag{66}$$

$$\varepsilon_{33} = \frac{2}{3}\varepsilon \tag{67}$$

$$\Phi_{33_1} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_3u_3} - \frac{2}{3}k\right) \tag{68}$$

$$\Phi_{33_2} = -C_{R2}\frac{2}{3}\overline{u_1u_2}S_s \tag{69}$$

$$\Phi_{33} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_3u_3} - \frac{2}{3}k\right) - C_{R2}\frac{2}{3}\overline{u_1u_2}S_s \tag{70}$$

$$\boxed{\frac{\partial\overline{u_3u_3}}{\partial t} = -\frac{2}{3}\varepsilon - C_{R1}\frac{\varepsilon}{k}\left(\overline{u_3u_3} - \frac{2}{3}k\right) - C_{R2}\frac{2}{3}\overline{u_1u_2}S_s} \tag{71}$$

$\underline{i=1 \quad j=2:}$

$$P_{12} = -\overline{u_2u_2}S_s \tag{72}$$

$$\varepsilon_{12} = 0 \tag{73}$$

$$\Phi_{12_1} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_1u_2}\right) \tag{74}$$

$$\Phi_{12_2} = C_{R2}\overline{u_2u_2}S_s \tag{75}$$

$$\Phi_{12} = -C_{R1}\frac{\varepsilon}{k}\left(\overline{u_1u_2}\right) + C_{R2}\overline{u_2u_2}S_s \tag{76}$$

$$\boxed{\frac{\partial\overline{u_1u_2}}{\partial t} = -\overline{u_2u_2}S_s - C_{R1}\frac{\varepsilon}{k}\overline{u_1u_2} + C_{R2}\overline{u_2u_2}S_s} \tag{77}$$

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

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

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 ($\mathbf{RSM_R}$)

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:

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

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

Lumley Reynolds Stress Model ($\mathbf{RSM_L}$)

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:

$$\boxed{\frac{\partial \varepsilon}{\partial t} = \sqrt{2} C_1 \mathcal{S} \varepsilon - C_2 \frac{\varepsilon^2}{k}} \tag{80}$$

coupled with the new $\mathcal{S}$ equation:

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

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 & McComb[5] (2018):

$$
\begin{aligned}
Re_{L_0} &= 3828.2 \\
Re_{\lambda_0} &= 353.7 \\
\nu_0 &= 0.0002 \quad \left(\frac{m^2}{s}\right) \\
\tau_0 &= 1.94 \quad (s) \\
S_s &= 0 \quad \left(\frac{1}{s}\right) \\
U_0 &= \sqrt{\frac{Re_{L_0}\nu_0}{\tau_0}} = 0.628 \quad \left(\frac{m}{s}\right)
\end{aligned}
$$

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

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

"C128W" Case Parameters

Rogers[6] (1986) provides the following parameters and initial conditions:

$$k_0 = 14.7$$
$$\varepsilon_0 = 155$$
$$S_s = 56.568$$
$$S_s t_0 = 12$$

Therefore:

$$t_0 = \frac{S_s t_0}{S_s} = 0.212$$
$$\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}\left(k_0\right) = 9.8$$
$$\overline{uv}_0 = 0$$

Finally, we choose:

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

because $\tau_0 = \dfrac{k_0}{\varepsilon_0} = 0.095 << t_0 = 0.212$.

"C128U" Case Parameters

Rogers[6] (1986) provides the following parameters and initial conditions:

$$\begin{aligned} k_0 &= 6.38 \\ \varepsilon_0 &= 41 \\ S_s &= 28.284 \\ S_s t_0 &= 8 \end{aligned}$$

Therefore:

$$\begin{aligned} t_0 &= \frac{S_s t_0}{S_s} = 0.283 \\ \overline{uu}_0 &= \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}\big(k_0\big) = 4.25 \\ \overline{uv}_0 &= 0 \end{aligned}$$

Finally, we choose:

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

because $\tau_0 = \dfrac{k_0}{\varepsilon_0} = 0.156 << t_0 = 0.283$.

"Case2" Case Parameters

Matsumoto[7] (1994) provides the following parameters and initial conditions:

$$\begin{aligned} k_0 &= 0.252 \\ \varepsilon_0 &= 0.234 \\ S_s &= 10\sqrt{2} \\ S_s t_0 &= 2 \end{aligned}$$

Therefore:

$$\begin{aligned} t_0 &= \frac{S_s t_0}{S_s} = 0.141 \\ \overline{uu}_0 &= \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}\big(k_0\big) = .168 \\ \overline{uv}_0 &= 0 \end{aligned}$$

Finally, we choose:

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

because $\tau_0 = \dfrac{k_0}{\varepsilon_0} = 1.08 >> t_0 = 0.141$.

"HSh" Case Parameters

Lee[8] (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:

$$q_0^2 = 2(k_0) = 2 \quad \left(\frac{m^2}{s^2}\right)$$
$$\varepsilon_0 = \frac{(q_0^2)^2}{\nu(Re_{T_0})} = 325.2 \quad \left(\frac{m^2}{s^3}\right)$$
$$S_s = \eta_0 \frac{\varepsilon_0}{k_0} = 5451 \quad \left(\frac{1}{s}\right)$$
$$\overline{uu}_0 = \overline{vv}_0 = \overline{ww}_0 = \frac{2}{3}\left(k_0\right) = .666 \quad \left(\frac{m^2}{s^2}\right)$$
$$\overline{uv}_0 = 0 \quad \left(\frac{m^2}{s^2}\right)$$

Finally, we choose:

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

"ω_{Low}" Case Parameters

Yu & Girimaji[9] (2006) provide the following constraints:

$$S_s = S_{max} \sin\left(\omega t\right)$$
$$S_0^* = S_{max} \frac{k_0}{\varepsilon_0} = 3.3$$
$$\frac{\omega}{S_{max}} = 0.25$$
$$S_{max} = 0.0006875$$

Arbitrarily choose $k_0 = 1$. Therefore:

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

Finally, we choose:

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

"ω_{Med}" Case Parameters

Yu & Girimaji[9] (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:

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

Finally, we choose:

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

"ω_{High}" Case Parameters

Yu & Girimaji[9] (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:

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

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.

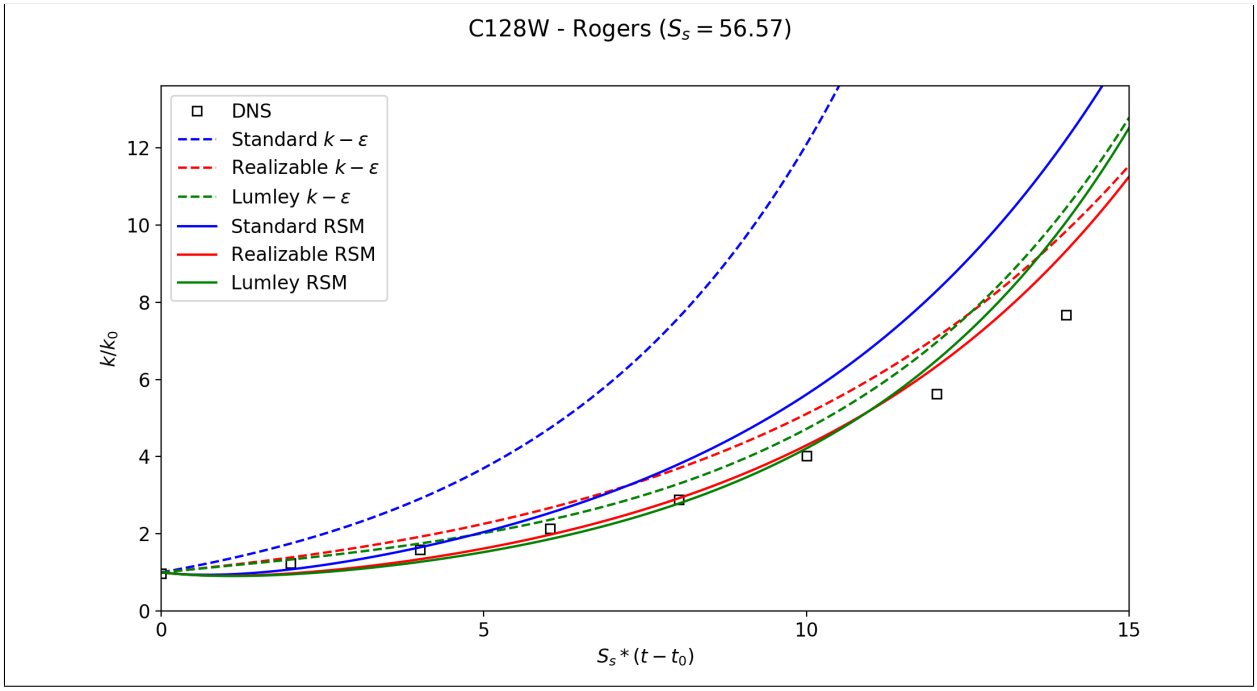

Figure 1: C128W Results

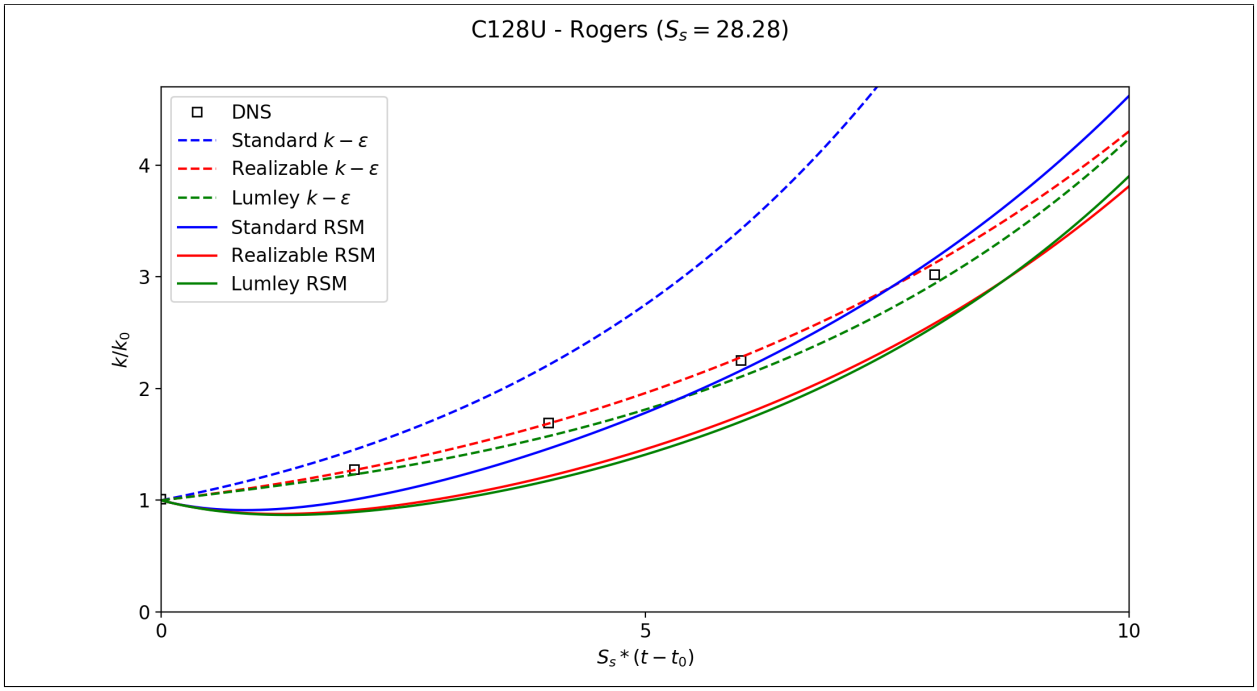

Figure 2: C128U Results

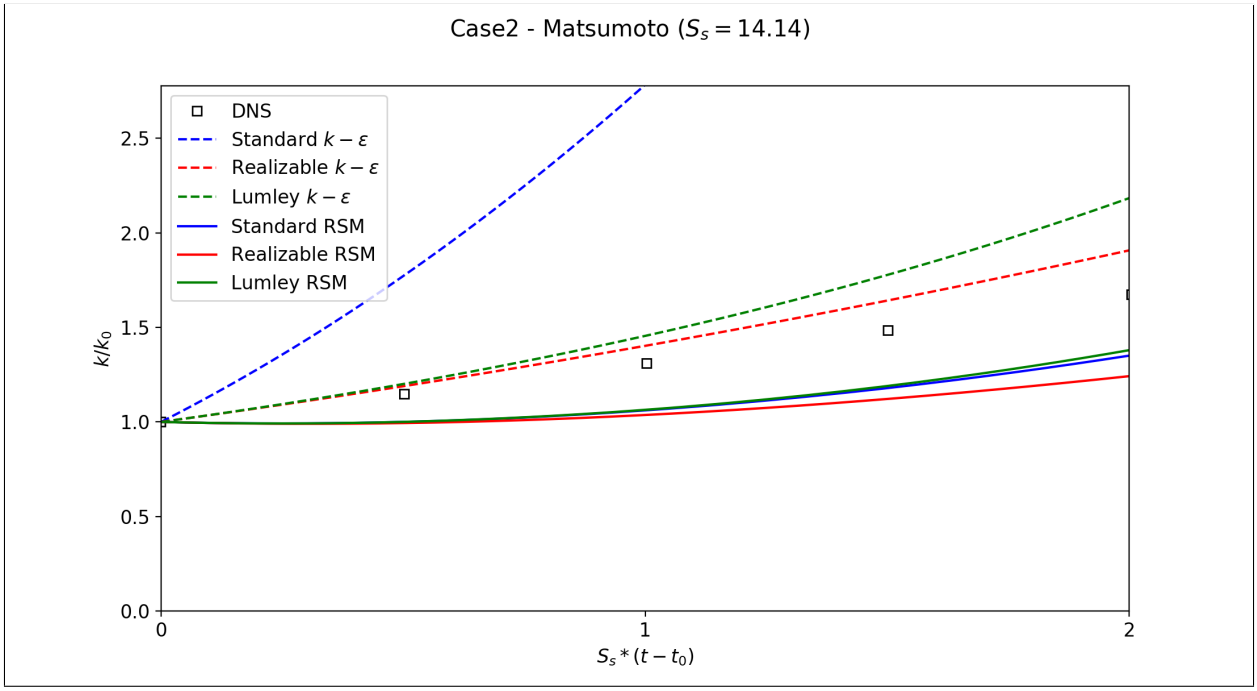

Figure 3: Case2 Results

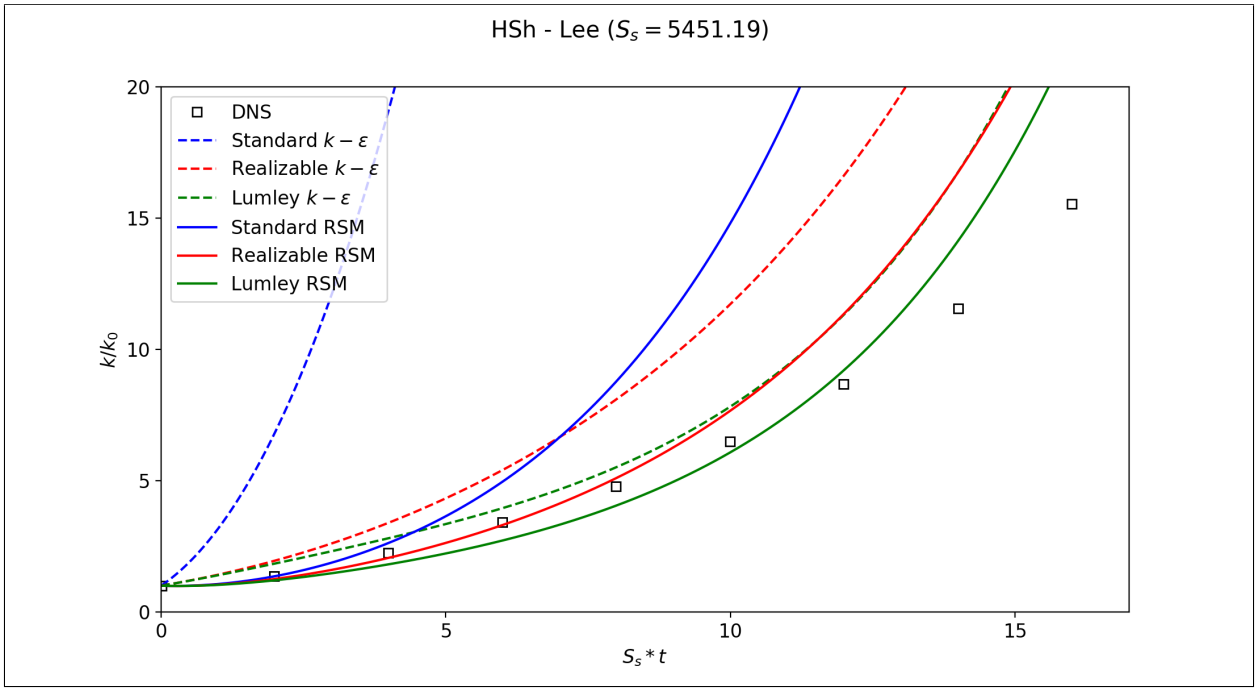

Figure 4: HSh Results

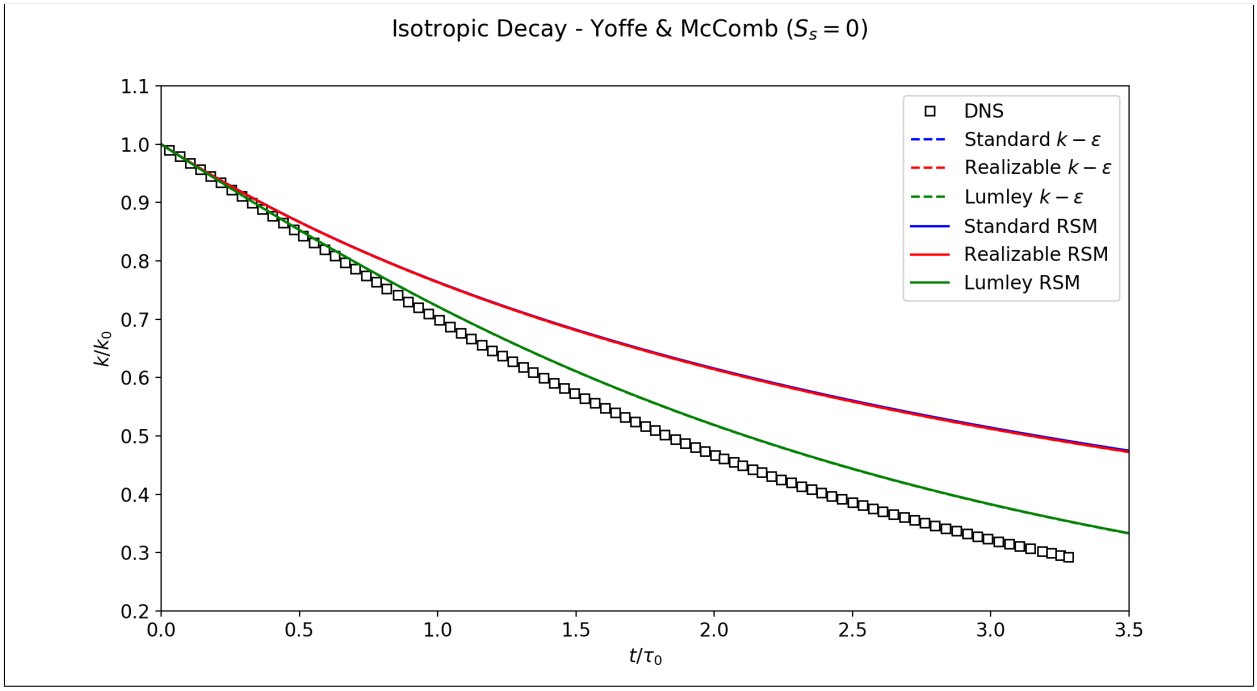

Figure 5: Decay Results

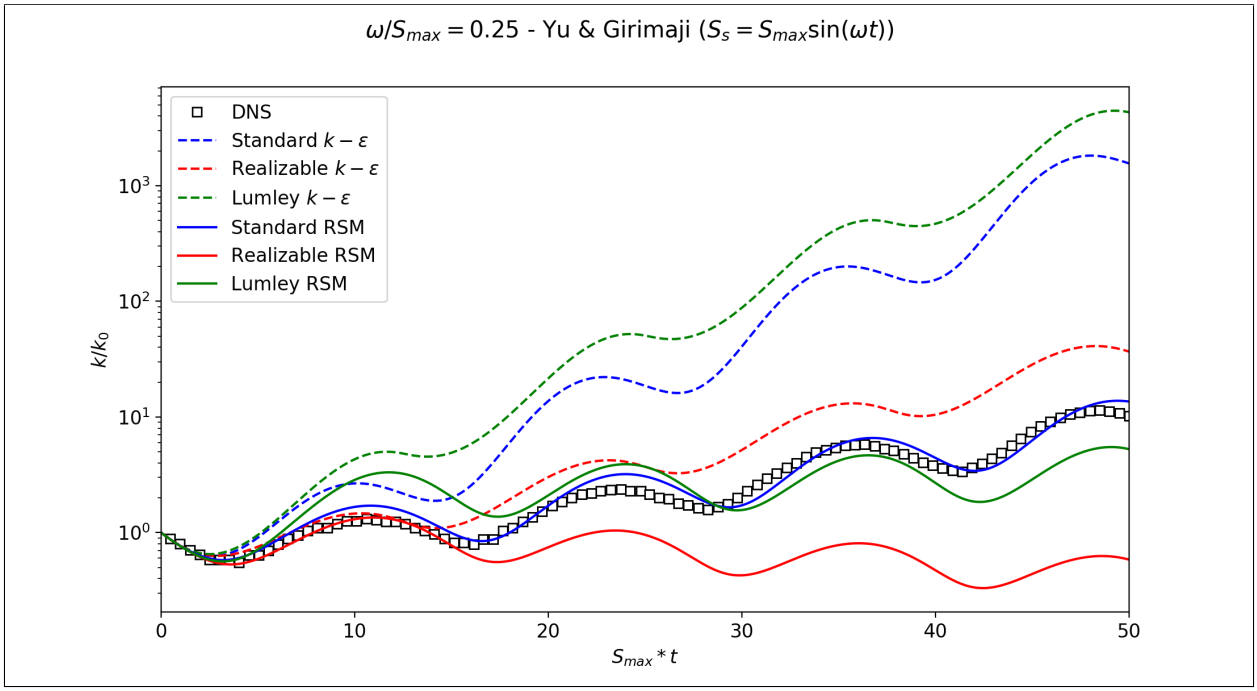

Figure 6: ω_{Low} Results

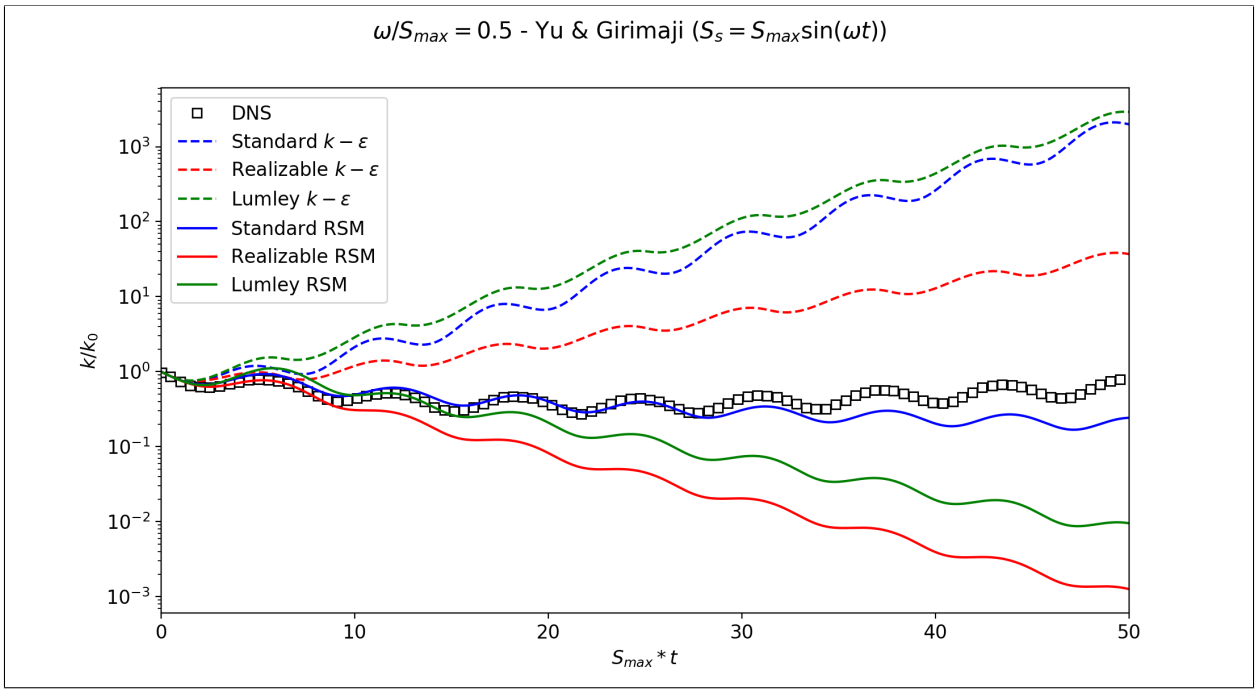

Figure 7: ω_{Med} Results

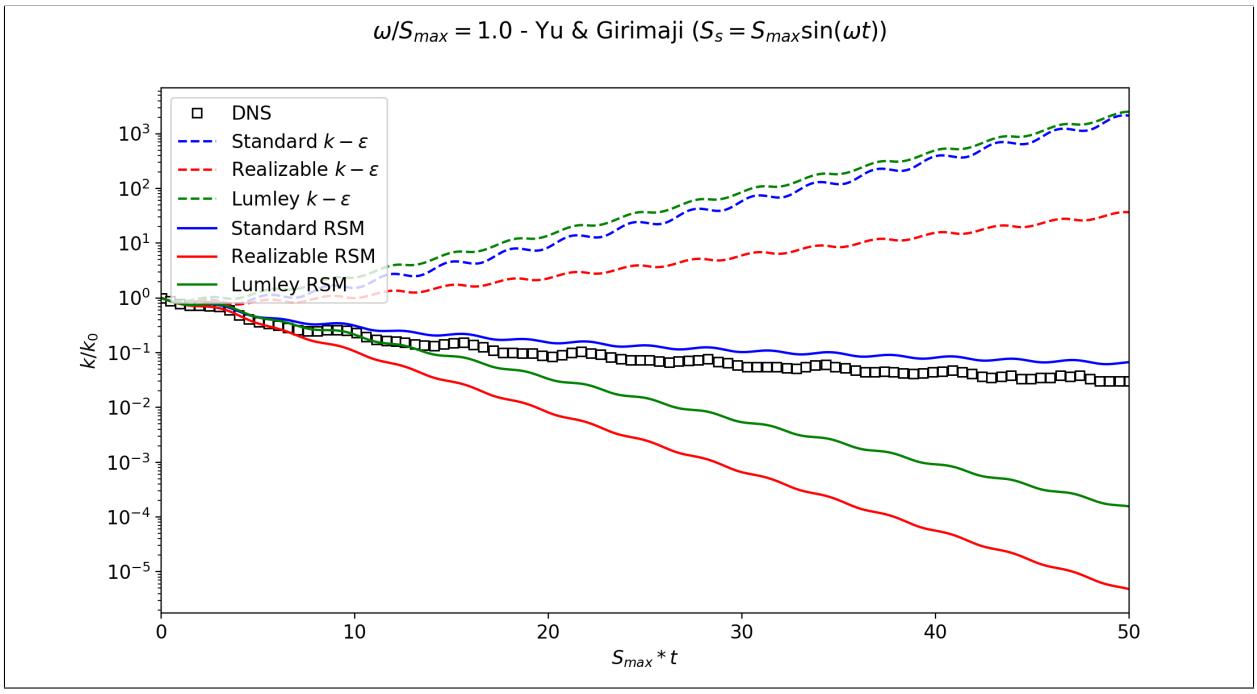

Figure 8: ω_{High} Results

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 isotopic turbulent delay, 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 opprotunity 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)

$\mathcal{S}$ - 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:10.1017s0022112006001832

Appendix III: Python Code

```
#!/usr/bin/env python

#Homogeneous Shear Flow Turbulence Model Comparison
#Models Compared: Standard k-eps, Realizable k-eps, Lumley k-eps, Standard RSM, Realizable RSM, Lumley RSM
#Cases Considered: C128W (Rogers), C128U (Rogers), Case2 (Matsumoto), HSh (Lee), Decay (Yoffe & McComb), w_low (Yu

#Packages:
import numpy as np
import matplotlib.pyplot as mpl
import csv
import math

#Case Parameters#########################################################################

all_cases = ["C128W","C128U","Case2","HSh","Decay","w_low","w_med","w_high"]

#C128W Parameters:
k_0_C128W = 14.7
eps_0_C128W = 155
Sh_C128W = 56.568
Sht0_C128W = 12.0
t_0_C128W = Sht0_C128W/Sh_C128W
t_f_C128W = (15+Sht0_C128W)/Sh_C128W
S_0_C128W = Sh_C128W
sine_C128W = False
w_C128W = None

#C128U Parameters:
k_0_C128U = 6.38
eps_0_C128U = 41
Sh_C128U = 28.284
Sht0_C128U = 8.0
t_0_C128U = Sht0_C128U/Sh_C128U
t_f_C128U = (10+Sht0_C128U)/Sh_C128U
S_0_C128U = Sh_C128U
sine_C128U = False
w_C128U = None

#Case2 Parameters:
k_0_Case2 =  0.252
eps_0_Case2 = 0.234
Sh_Case2 = 10*(np.sqrt(2))
Sht0_Case2 = 2.0
t_0_Case2 = Sht0_Case2/Sh_Case2
t_f_Case2 = (2+Sht0_Case2)/Sh_Case2
S_0_Case2 = eps_0_Case2/k_0_Case2
sine_Case2 = False
w_Case2 = None

#HSh Parameters:
n_HSh = 16.7624
Re_T_HSh = 1230
k_0_HSh = 1
nu_HSh = 1E-5
q_0_2_HSh = 2*k_0_HSh
eps_0_HSh = q_0_2_HSh**2/nu_HSh/Re_T_HSh
Sh_HSh = n_HSh/(k_0_HSh/eps_0_HSh)
t_0_HSh = 0
t_f_HSh = (20/Sh_HSh)
S_0_HSh = n_HSh*eps_0_HSh/k_0_HSh
sine_HSh = False
w_HSh = None

#Decay Parameters:
nu_Decay = 0.0002
tau_0_Decay = 1.94
Re_lamda_0_Decay = 353.7
Re_L_0_Decay = 3828.2
U_0_Decay = np.sqrt(Re_L_0_Decay*nu_Decay/tau_0_Decay)
k_0_Decay = 3/2*U_0_Decay**2
eps_0_Decay = (U_0_Decay**2*np.sqrt(15)/np.sqrt(nu_Decay)/Re_lamda_0_Decay)**2
t_0_Decay = 0
t_f_Decay = 3.5*tau_0_Decay
Sh_Decay = 0
S_0_Decay = 1/tau_0_Decay
sine_Decay = False
w_Decay = None

#w_low Parameters:
S_star_w_low = 3.3
```

```
Sh_w_low = 0.0006875 #(Sh_max)
k_0_w_low = 1
eps_0_w_low = Sh_w_low*k_0_w_low/S_star_w_low
S_0_w_low = eps_0_w_low/k_0_w_low
t_0_w_low = 0/Sh_w_low
t_f_w_low = 50/Sh_w_low
sine_w_low = True
w_w_low = 0.25*Sh_w_low

#w_med Parameters:
S_star_w_med = 3.3
Sh_w_med = 0.0006875 #(Sh_max)
k_0_w_med = 1
eps_0_w_med = Sh_w_med*k_0_w_med/S_star_w_med
S_0_w_med = eps_0_w_med/k_0_w_med
t_0_w_med = 0/Sh_w_med
t_f_w_med = 50/Sh_w_med
sine_w_med = True
w_w_med = 0.5*Sh_w_med

#w_high Parameters:
S_star_w_high = 3.3
Sh_w_high = 0.0006875 #(Sh_max)
k_0_w_high = 1
eps_0_w_high = Sh_w_high*k_0_w_high/S_star_w_high
S_0_w_high = eps_0_w_high/k_0_w_high
t_0_w_high = 0/Sh_w_high
t_f_w_high = 50/Sh_w_high
sine_w_high = True
w_w_high = 1.0*Sh_w_high

#Model Constants####################################################################
n = 1.11

#Standard k-eps Constants:
C_mu_st = 0.09
C_eps1_st = 1.44
C_eps2_st = 1.92

#Realizable Constants:
A_0_real = 4
A_S_real = np.sqrt(6)*np.cos(np.pi/6)
C_2_real = (n+1)/n

#Lumley  Constants:
A_0_lum = 4
A_S_lum = np.sqrt(6)*np.cos(np.pi/6)
C_S1_lum = 1/n

#Standard Reynolds Stress Constants:
C_R1_rsms = 1.8
C_R2_rsms = 0.6
C_eps1_rsms = 1.44
C_eps2_rsms = 1.92

#Realizable Reynolds Stress Constants:
C_R1_rsmr = 1.8
C_R2_rsmr = 0.6
C_2_rsmr = (n+1)/n

#Lumley Reynolds Stress Constants:
C_R1_rsml = 1.8
C_R2_rsml = 0.6
C_S1_rsml = 1/n

#Read DNS Data #####################################################################
for v in all_cases:
    vars()['k_x_DNS_'+v+'_norm']=[]
    vars()['k_y_DNS_'+v+'_norm']=[]
    with open('/home/dtobin/Documents/NASA Research/DNS CSV Data/'+v+'_DNS.csv','r') as csvfile:
        data = csv.reader(csvfile,delimiter = ',')
        for row in data:
                eval('k_x_DNS_'+v+'_norm').append(float(row[0]))
                eval('k_y_DNS_'+v+'_norm').append(float(row[1]))
#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
k_y_DNS_C128W_norm = [x/k_0_C128W for x in k_y_DNS_C128W_norm]
k_y_DNS_C128U_norm = [x/k_0_C128U for x in k_y_DNS_C128U_norm]
k_y_DNS_Case2_norm = [x/k_0_Case2 for x in k_y_DNS_Case2_norm]

#Standard k-eps Solver#############################################################
def st_solver (C_mu,C_eps1,C_eps2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
```

```
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_st_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_st_dims = [k_0]
    eps_st_dims = [eps_0]
    Sh = []

    #Define Lambda Functions:
    F_k_st = lambda i,y : (C_mu*Sh[i]**2*y**2)/eps_st_dims[i]-eps_st_dims[i]
    F_eps_st = lambda i,y : ((C_eps1*C_mu_st*Sh[i]**2*(k_st_dims[i])**2)/y-C_eps2*y)*y/k_st_dims[i]

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_st_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(abs(Sh_max*math.sin(w*t_st_dims[i])))
        for x in ["k","eps"]:
            lfn = eval("F_"+x+"_st")
            var = eval(x+"_st_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)

    #Return Data for Plots
    return (k_st_dims,eps_st_dims,t_st_dims)

#Realizable k-eps Solver#############################################################
def real_solver (A_0,A_S,C_2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_real_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_real_dims = [k_0]
    eps_real_dims = [eps_0]
    Sh = []
    U_star = []
    eta = []
    C_1 = []
    C_mu = []

    #Define Lambda Functions:
    F_k_real = lambda i,y : (C_mu[i]*Sh[i]**2*y**2)/eps_real_dims[i]-eps_real_dims[i]
    F_eps_real = lambda i,y : (C_1[i]*abs(Sh[i])*y)-(C_2*y**2/k_real_dims[i])

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_real_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(Sh_max*math.sin(w*t_real_dims[i]))
        U_star.append(abs(Sh[i]))
        eta.append(abs(Sh[i])*k_real_dims[i]/eps_real_dims[i])
        C_1.append(max(0.43,eta[i]/(5+eta[i])))
        C_mu.append(1/(A_0+A_S*U_star[i]*(k_real_dims[i]/eps_real_dims[i])))
        for x in ["k","eps"]:
            lfn = eval("F_"+x+"_real")
            var = eval(x+"_real_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)

    #Return Data for Plots
    return (k_real_dims,eps_real_dims,t_real_dims)

#Lumley k-eps Solver#################################################################
def lum_solver (A_0,A_S,C_S1,k_0,eps_0,S_0,t_0,t_f,Sh_max,sine,w):
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_lum_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_lum_dims = [k_0]
    eps_lum_dims = [eps_0]
```

```
    S_lum_dims = [S_0]
    Sh = []
    U_star = []
    eta = []
    C_1 = []
    C_2 = []
    C_mu = []

    #Define Lambda Functions:
    F_k_lum = lambda i,y : (C_mu[i]*Sh[i]**2*y**2)/eps_lum_dims[i]-eps_lum_dims[i]
    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])
    F_S_lum = lambda i,y : (abs(Sh[i])/np.sqrt(2)-y)*(C_S1*eps_lum_dims[i]/k_lum_dims[i])

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_lum_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(Sh_max*math.sin(w*t_lum_dims[i]))
        U_star.append(abs(Sh[i]))
        eta.append(abs(Sh[i])*k_lum_dims[i]/eps_lum_dims[i])
        C_1.append(max(0.43,eta[i]/(5+eta[i])))
        C_2.append(((n+1)/n)+C_1[i])
        C_mu.append(1/(A_0+A_S*U_star[i]*(k_lum_dims[i]/eps_lum_dims[i])))
        for x in ["k","eps","S"]:
            lfn = eval("F_"+x+"_lum")
            var = eval(x+"_lum_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)

    #Return Data for Plots
    return (k_lum_dims,eps_lum_dims,S_lum_dims,t_lum_dims)

#Standard RSM Solver##################################################################
def rsms_solver (C_R1,C_R2,C_eps1,C_eps2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_rsms_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_rsms_dims = [k_0]
    eps_rsms_dims = [eps_0]
    uu_rsms_dims = [2*k_0/3]
    vv_rsms_dims = [2*k_0/3]
    ww_rsms_dims = [2*k_0/3]
    uv_rsms_dims = [0]
    Sh = []
    b11 = []
    b22 = []

    #Define Lambda Functions:
    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]
    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
    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]
    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]
    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

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_rsms_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(Sh_max*math.sin(w*t_rsms_dims[i]))
        for x in ["eps","uu","vv","ww","uv"]:
            lfn = eval("F_"+x+"_rsms")
            var = eval(x+"_rsms_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
        k_rsms_dims.append(0.5*(uu_rsms_dims[i+1]+vv_rsms_dims[i+1]+ww_rsms_dims[i+1]))
        b11.append(uu_rsms_dims[i]/2/k_rsms_dims[i]-(1/3))
        b22.append(vv_rsms_dims[i]/2/k_rsms_dims[i]-(1/3))

    #Return Data for Plots
    return (k_rsms_dims,eps_rsms_dims,t_rsms_dims,b11,b22)

```

```
#Realizable RSM Solver#################################################################
def rsmr_solver (C_R1,C_R2,C_2,k_0,eps_0,t_0,t_f,Sh_max,sine,w):
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_rsmr_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_rsmr_dims = [k_0]
    eps_rsmr_dims = [eps_0]
    uu_rsmr_dims = [2*k_0/3]
    vv_rsmr_dims = [2*k_0/3]
    ww_rsmr_dims = [2*k_0/3]
    uv_rsmr_dims = [0]
    Sh = []
    eta = []
    C_1 = []

    #Define Lambda Functions:
    F_eps_rsmr = lambda i,y : (C_1[i]*abs(Sh[i])*y)-(C_2*y**2/k_rsmr_dims[i])
    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
    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]
    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]
    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

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_rsmr_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(Sh_max*math.sin(w*t_rsmr_dims[i]))
        eta.append(abs(Sh[i])*k_rsmr_dims[i]/eps_rsmr_dims[i])
        C_1.append(max(0.43,eta[i]/(5+eta[i])))
        for x in ["eps","uu","vv","ww","uv"]:
            lfn = eval("F_"+x+"_rsmr")
            var = eval(x+"_rsmr_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
        k_rsmr_dims.append(0.5*(uu_rsmr_dims[i+1]+vv_rsmr_dims[i+1]+ww_rsmr_dims[i+1]))

    #Return Data for Plots
    return (k_rsmr_dims,eps_rsmr_dims,t_rsmr_dims)

#Lumley RSM Solver#####################################################################
def rsml_solver (C_R1,C_R2,C_S1,k_0,eps_0,S_0,t_0,t_f,Sh_max,sine,w):
    #Time Domain:
    dt = (t_f-t_0)/1e3

    #Initialization:
    t_rsml_dims = np.linspace(t_0,t_f,(t_f-t_0)/dt)
    k_rsml_dims = [k_0]
    eps_rsml_dims = [eps_0]
    S_rsml_dims = [S_0]
    uu_rsml_dims = [2*k_0/3]
    vv_rsml_dims = [2*k_0/3]
    ww_rsml_dims = [2*k_0/3]
    uv_rsml_dims = [0]
    Sh = []
    eta = []
    C_1 = []
    C_2 = []

    #Define Lambda Functions:
    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])
    F_S_rsml = lambda i,y : (abs(Sh[i])/np.sqrt(2)-y)*(C_S1*eps_rsml_dims[i]/k_rsml_dims[i])
    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
    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]
    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]
    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

    #4th Order Runge-Kutta Method:
    for i in np.arange(0,len(t_rsml_dims)-1):
        if sine is False:
            Sh.append(Sh_max)
        else:
            Sh.append(Sh_max*math.sin(w*t_rsml_dims[i]))
        eta.append(abs(Sh[i])*k_rsml_dims[i]/eps_rsml_dims[i])
        C_1.append(max(0.43,eta[i]/(5+eta[i])))
        C_2.append(((n+1)/n)+C_1[i])
```

```
        for x in ["eps","S","uu","vv","ww","uv"]:
            lfn = eval("F_"+x+"_rsml")
            var = eval(x+"_rsml_dims")
            K1 = lfn(i,var[i])
            K2 = lfn(i,var[i] + 0.5*K1*dt)
            K3 = lfn(i,var[i] + 0.5*K2*dt)
            K4 = lfn(i,var[i] + K3*dt)
            var.append(var[i] + (1/6)*(K1+2*K2+2*K3+K4)*dt)
        k_rsml_dims.append(0.5*(uu_rsml_dims[i+1]+vv_rsml_dims[i+1]+ww_rsml_dims[i+1]))

    #Return Data for Plots
    return (k_rsml_dims,eps_rsml_dims,S_rsml_dims,t_rsml_dims)

#Standard k-eps Data#################################################################

#Call St Function for Each Case and Create Dim. and Norm. k and eps:
k_st_dims_dict = {}
eps_st_dims_dict = {}
t_st_dims_dict = {}
k_st_norm_dict = {}
eps_st_norm_dict = {}
for z in all_cases:
    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
    k_st_dims_dict[z] = st_res[0]
    eps_st_dims_dict[z] = st_res[1]
    t_st_dims_dict[z] = st_res[2]
    k_st_norm_dict[z] = [x/eval("k_0_"+z) for x in k_st_dims_dict[z]]
    eps_st_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_st_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_st_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_st_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_st_dims_dict[z]]
#Normalize t to Sh(t)
for z in ["HSh"]:
    t_st_norm_dict[z] = [eval("Sh_"+z)*x for x in t_st_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_st_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_st_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_st_norm_dict[z] = [eval("Sh_"+z)*x for x in t_st_dims_dict[z]]

#Realizable k-eps Data###############################################################

#Call Real Function for Each Case and Create Dim. and Norm. k and eps:
k_real_dims_dict = {}
eps_real_dims_dict = {}
t_real_dims_dict = {}
k_real_norm_dict = {}
eps_real_norm_dict = {}
for z in all_cases:
    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)
    k_real_dims_dict[z] = real_res[0]
    eps_real_dims_dict[z] = real_res[1]
    t_real_dims_dict[z] = real_res[2]
    k_real_norm_dict[z] = [x/eval("k_0_"+z) for x in k_real_dims_dict[z]]
    eps_real_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_real_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_real_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_real_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_real_dims_dict[z]]
#Normalize t to Sh(t)
for z in ["HSh"]:
    t_real_norm_dict[z] = [eval("Sh_"+z)*x for x in t_real_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_real_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_real_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_real_norm_dict[z] = [eval("Sh_"+z)*x for x in t_real_dims_dict[z]]

#Lumley k-eps Data###################################################################

#Call Lum Function for Each Case and Create Dim. and Norm. k,eps, and S:
k_lum_dims_dict = {}
eps_lum_dims_dict = {}
S_lum_dims_dict = {}
t_lum_dims_dict = {}
k_lum_norm_dict = {}
```

```
eps_lum_norm_dict = {}
S_lum_norm_dict = {}
for z in all_cases:
    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
    k_lum_dims_dict[z] = lum_res[0]
    eps_lum_dims_dict[z] = lum_res[1]
    S_lum_dims_dict[z] = lum_res[2]
    t_lum_dims_dict[z] = lum_res[3]
    k_lum_norm_dict[z] = [x/eval("k_0_"+z) for x in k_lum_dims_dict[z]]
    eps_lum_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_lum_dims_dict[z]]
    S_lum_norm_dict[z] = [x/eval("S_0_"+z) for x in S_lum_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_lum_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_lum_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_lum_dims_dict[z]]
#Normalize t to Sh(t)
for z in ["HSh"]:
    t_lum_norm_dict[z] = [eval("Sh_"+z)*x for x in t_lum_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_lum_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_lum_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_lum_norm_dict[z] = [eval("Sh_"+z)*x for x in t_lum_dims_dict[z]]

#Standard RSM Data#####################################################################

#Call RSMS Function for Each Case and Create Dim. and Norm. k and eps:
k_rsms_dims_dict = {}
eps_rsms_dims_dict = {}
t_rsms_dims_dict = {}
k_rsms_norm_dict = {}
eps_rsms_norm_dict = {}
b11_rsms_dims_dict = {}
b22_rsms_dims_dict = {}
for z in all_cases:
    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_"+
    k_rsms_dims_dict[z] = rsms_res[0]
    eps_rsms_dims_dict[z] = rsms_res[1]
    t_rsms_dims_dict[z] = rsms_res[2]
    b11_rsms_dims_dict[z] = rsms_res[3]
    b22_rsms_dims_dict[z] = rsms_res[4]
    k_rsms_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsms_dims_dict[z]]
    eps_rsms_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsms_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_rsms_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_rsms_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsms_dims_dict[z]]
#Normalize t to Sh(t)
for z in ["HSh"]:
    t_rsms_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsms_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_rsms_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsms_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_rsms_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsms_dims_dict[z]]

#Realizable RSM Data###################################################################

#Call RSMR Function for Each Case and Create Dim. and Norm. k and eps:
k_rsmr_dims_dict = {}
eps_rsmr_dims_dict = {}
t_rsmr_dims_dict = {}
k_rsmr_norm_dict = {}
eps_rsmr_norm_dict = {}
for z in all_cases:
    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_"+
    k_rsmr_dims_dict[z] = rsmr_res[0]
    eps_rsmr_dims_dict[z] = rsmr_res[1]
    t_rsmr_dims_dict[z] = rsmr_res[2]
    k_rsmr_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsmr_dims_dict[z]]
    eps_rsmr_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsmr_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_rsmr_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsmr_dims_dict[z]]
#Normalize t to Sh(t)
```

```
for z in ["HSh"]:
    t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsmr_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_rsmr_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsmr_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_rsmr_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsmr_dims_dict[z]]

#Lumley RSM Data#####################################################################

#Call RSML Function for Each Case and Create Dim. and Norm. k, eps, and S:
k_rsml_dims_dict = {}
eps_rsml_dims_dict = {}
S_rsml_dims_dict ={}
t_rsml_dims_dict = {}
k_rsml_norm_dict = {}
eps_rsml_norm_dict = {}
S_rsml_norm_dict = {}
for z in all_cases:
    rsml_res = rsml_solver(C_R1_rsml,C_R2_rsml,C_S1_rsml,eval("k_0_"+z),eval("eps_0_"+z),eval("S_0_"+z),eval("t_0_"
    k_rsml_dims_dict[z] = rsml_res[0]
    eps_rsml_dims_dict[z] = rsml_res[1]
    S_rsml_dims_dict[z] = rsml_res[2]
    t_rsml_dims_dict[z] = rsml_res[3]
    k_rsml_norm_dict[z] = [x/eval("k_0_"+z) for x in k_rsml_dims_dict[z]]
    eps_rsml_norm_dict[z] = [x/eval("eps_0_"+z) for x in eps_rsml_dims_dict[z]]
    S_rsml_norm_dict[z] = [x/eval("S_0_"+z) for x in S_rsml_dims_dict[z]]

#Normalize t to Sh(t-t0)
t_rsml_norm_dict = {}
for z in ["C128W","C128U","Case2"]:
    t_rsml_norm_dict[z] = [eval("Sh_"+z)*x-eval("Sht0_"+z) for x in t_rsml_dims_dict[z]]
#Normalize t to Sh(t)
for z in ["HSh"]:
    t_rsml_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsml_dims_dict[z]]
#Normalize t to t/tau0
for z in ["Decay"]:
    t_rsml_norm_dict[z] = [x/eval("tau_0_"+z) for x in t_rsml_dims_dict[z]]
#Normalize t to Sh_max*t
for z in ["w_low","w_med","w_high"]:
    t_rsml_norm_dict[z] = [eval("Sh_"+z)*x for x in t_rsml_dims_dict[z]]

#CREATE PLOTS:#######################################################################
plot_L = 11
plot_W = 6
plot_DPI = 200
mpl.rcParams.update({'font.size': 12})

##############################C128W NORMALIZED PLOTS:##############################
mpl.figure(1)
mpl.suptitle('C128W - Rogers $(S_s = '+str(round(Sh_C128W,2))+')$')

#mpl.subplot(1,2,1)
#DNS:
mpl.plot (k_x_DNS_C128W_norm,k_y_DNS_C128W_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.plot (t_st_norm_dict['C128W'],k_st_norm_dict['C128W'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.plot (t_real_norm_dict['C128W'],k_real_norm_dict['C128W'],'--',color = 'red',label='Realizable $k-\\epsilon$')
#Lumley k-eps:
mpl.plot (t_lum_norm_dict['C128W'],k_lum_norm_dict['C128W'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.plot (t_rsms_norm_dict['C128W'],k_rsms_norm_dict['C128W'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.plot (t_rsmr_norm_dict['C128W'],k_rsmr_norm_dict['C128W'],color = 'red',label='Realizable RSM')
#RSML:
mpl.plot (t_rsml_norm_dict['C128W'],k_rsml_norm_dict['C128W'],color = 'green',label='Lumley RSM')

#Formating(1,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_s*(t-t_0)$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,15)
mpl.ylim (0,200/k_0_C128W)
mpl.xticks (np.linspace(0,15,4))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['C128W'],S_lum_norm_dict['C128W'],'--',color = 'green',label='Lumley $k-\\epsilon$')
```

```
# #Target Value:
# mpl.plot(t_lum_norm_dict['C128W'],[Sh_C128W/S_0_C128W for x in t_lum_norm_dict['C128W']],'-',color='black',label=

# #Formating(1,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_s*(t-t_0)$")
# mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
# mpl.xlim (0,15)
# mpl.xticks (np.linspace(0,15,4))
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/C128W.png',dpi = plot_DPI)

################################C128U NORMALIZED PLOTS:##############################
mpl.figure(2)
mpl.suptitle('C128U - Rogers $(S_s = '+str(round(Sh_C128U,2))+')$')

#mpl.subplot(1,2,1)
#DNS:
mpl.plot (k_x_DNS_C128U_norm,k_y_DNS_C128U_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.plot (t_st_norm_dict['C128U'],k_st_norm_dict['C128U'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.plot (t_real_norm_dict['C128U'],k_real_norm_dict['C128U'],'--',color = 'red',label='Realizable $k-\\epsilon$')
#Lumley k-eps:
mpl.plot (t_lum_norm_dict['C128U'],k_lum_norm_dict['C128U'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.plot (t_rsms_norm_dict['C128U'],k_rsms_norm_dict['C128U'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.plot (t_rsmr_norm_dict['C128U'],k_rsmr_norm_dict['C128U'],color = 'red',label='Realizable RSM')
#RSML:
mpl.plot (t_rsml_norm_dict['C128U'],k_rsml_norm_dict['C128U'],color = 'green',label='Lumley RSM')

#Formating(2,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_s*(t-t_0)$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,10)
mpl.ylim (0,30/k_0_C128U)
mpl.xticks (np.linspace(0,10,3))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['C128U'],S_lum_norm_dict['C128U'],'--',color = 'green',label='Lumley $k-\\epsilon$')
# #Target Value:
# mpl.plot(t_lum_norm_dict['C128U'],[Sh_C128U/S_0_C128U for x in t_lum_norm_dict['C128U']],'-',color='black',label=

# #Formating(2,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_s*(t-t_0)$")
# mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
# mpl.xlim (0,10)
# mpl.xticks (np.linspace(0,10,3))
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/C128U.png',dpi = plot_DPI)

################################Case2 NORMALIZED PLOTS:##############################
mpl.figure(3)
mpl.suptitle('Case2 - Matsumoto $(S_s = '+str(round(Sh_Case2,2))+')$')

#mpl.subplot(1,2,1)
#DNS:
mpl.plot (k_x_DNS_Case2_norm,k_y_DNS_Case2_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.plot (t_st_norm_dict['Case2'],k_st_norm_dict['Case2'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.plot (t_real_norm_dict['Case2'],k_real_norm_dict['Case2'],'--',color = 'red',label='Realizable $k-\\epsilon$')
#Lumley k-eps:
mpl.plot (t_lum_norm_dict['Case2'],k_lum_norm_dict['Case2'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.plot (t_rsms_norm_dict['Case2'],k_rsms_norm_dict['Case2'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.plot (t_rsmr_norm_dict['Case2'],k_rsmr_norm_dict['Case2'],color = 'red',label='Realizable RSM')
```

```
#RSML:
mpl.plot (t_rsml_norm_dict['Case2'],k_rsml_norm_dict['Case2'],color = 'green',label='Lumley RSM')

#Formating(3,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_s*(t-t_0)$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,2)
mpl.ylim (0,0.7/k_0_Case2)
mpl.xticks (np.linspace(0,2,3))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['Case2'],S_lum_norm_dict['Case2'],'--',color = 'green',label='Lumley $k-\\epsilon$')
# #Target Value:
# mpl.plot(t_lum_norm_dict['Case2'],[Sh_Case2/S_0_Case2 for x in t_lum_norm_dict['Case2']],'-',color='black',label=

# #Formating(3,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_s*(t-t_0)$")
# mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
# mpl.xlim (0,2)
# mpl.xticks (np.linspace(0,2,3))
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/Case2.png',dpi = plot_DPI)

##################################HSh NORMALIZED PLOTS:############################
mpl.figure(4)
mpl.suptitle('HSh - Lee $(S_s = '+str(round(Sh_HSh,2))+')$')

#mpl.subplot(1,2,1)
#DNS:
mpl.plot (k_x_DNS_HSh_norm,k_y_DNS_HSh_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.plot (t_st_norm_dict['HSh'],k_st_norm_dict['HSh'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.plot (t_real_norm_dict['HSh'],k_real_norm_dict['HSh'],'--',color = 'red',label='Realizable $k-\\epsilon$')
#Lumley k-eps:
mpl.plot (t_lum_norm_dict['HSh'],k_lum_norm_dict['HSh'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.plot (t_rsms_norm_dict['HSh'],k_rsms_norm_dict['HSh'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.plot (t_rsmr_norm_dict['HSh'],k_rsmr_norm_dict['HSh'],color = 'red',label='Realizable RSM')
#RSML:
mpl.plot (t_rsml_norm_dict['HSh'],k_rsml_norm_dict['HSh'],color = 'green',label='Lumley RSM')

#Formating(4,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_s*t$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,17)
mpl.ylim (0,20)
mpl.xticks ([0,5,10,15])
mpl.yticks(np.linspace(0,20,5))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['HSh'],S_lum_norm_dict['HSh'],'--',color = 'green',label='Lumley $k-\\epsilon$')
# #Target Value:
# mpl.plot(t_lum_norm_dict['HSh'],[Sh_HSh/S_0_HSh for x in t_lum_norm_dict['HSh']],'-',color='black',label='Target

# #Formating(4,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_s*t$")
# mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
# mpl.xlim (0,17)
# mpl.xticks ([0,5,10,15])
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/HSh.png',dpi = plot_DPI)

##################################Decay NORMALIZED PLOTS:############################
```

```
mpl.figure(5)
mpl.suptitle('Isotropic Decay - Yoffe & McComb $(S_s = '+str(round(Sh_Decay,2))+')$')

#mpl.subplot(1,2,1)
#DNS:
mpl.plot (k_x_DNS_Decay_norm,k_y_DNS_Decay_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.plot (t_st_norm_dict['Decay'],k_st_norm_dict['Decay'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.plot (t_real_norm_dict['Decay'],k_real_norm_dict['Decay'],'--',color = 'red',label='Realizable $k-\\epsilon$')
#Lumley k-eps:
mpl.plot (t_lum_norm_dict['Decay'],k_lum_norm_dict['Decay'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.plot (t_rsms_norm_dict['Decay'],k_rsms_norm_dict['Decay'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.plot (t_rsmr_norm_dict['Decay'],k_rsmr_norm_dict['Decay'],color = 'red',label='Realizable RSM')
#RSML:
mpl.plot (t_rsml_norm_dict['Decay'],k_rsml_norm_dict['Decay'],color = 'green',label='Lumley RSM')

#Formating(5,1):
mpl.legend(loc='upper right',ncol=1)
mpl.xlabel("$t/\\tau_0$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,3.5)
mpl.ylim (0.2,1.1)
mpl.xticks (np.linspace(0,3.5,8))
mpl.yticks(np.linspace(0.2,1.1,10))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['Decay'],S_lum_norm_dict['Decay'],'--',color = 'green',label='Lumley $k-\\epsilon$')
# #Target Value:
# mpl.plot(t_lum_norm_dict['Decay'],[Sh_Decay/S_0_Decay for x in t_lum_norm_dict['Decay']],'-',color='black',label=

# #Formating(5,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$t/\\tau_0$")
# mpl.ylabel("$\\mathcal{S}/\\mathcal{S}_0$")
# mpl.xlim (0,3.5)
# mpl.xticks (np.linspace(0,3.5,8))
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/Decay.png',dpi = plot_DPI)

################################w_low NORMALIZED PLOTS:############################
mpl.figure(6)
mpl.suptitle('$\\omega/S_{max} = 0.25$ - Yu & Girimaji ($S_s = S_{max}\\sin{(\\omega t)}$)')

#mpl.subplot(1,2,1)
#DNS:
mpl.semilogy (k_x_DNS_w_low_norm,k_y_DNS_w_low_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.semilogy (t_st_norm_dict['w_low'],k_st_norm_dict['w_low'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.semilogy (t_real_norm_dict['w_low'],k_real_norm_dict['w_low'],'--',color = 'red',label='Realizable $k-\\epsilon
#Lumley k-eps:
mpl.semilogy (t_lum_norm_dict['w_low'],k_lum_norm_dict['w_low'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.semilogy (t_rsms_norm_dict['w_low'],k_rsms_norm_dict['w_low'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.semilogy (t_rsmr_norm_dict['w_low'],k_rsmr_norm_dict['w_low'],color = 'red',label='Realizable RSM')
#RSML:
mpl.semilogy (t_rsml_norm_dict['w_low'],k_rsml_norm_dict['w_low'],color = 'green',label='Lumley RSM')

#Formating(6,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_{max}*t$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,50)
#mpl.ylim (0,20)
mpl.xticks (np.linspace(0,50,6))
#mpl.yticks(np.linspace(0,20,3))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['w_low'],[x/Sh_w_low for x in S_lum_dims_dict['w_low']],'--',color = 'green',label=' $\
```

```
# #Sh:
# 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',lab

# #Formating(6,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_{max}*t$")
# mpl.xlim (0,50)
# mpl.xticks (np.linspace(0,50,6))
# mpl.ylim (-3,3)
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_low.png',dpi = plot_DPI)

##############################w_med NORMALIZED PLOTS:############################
mpl.figure(7)
mpl.suptitle('$\\omega/S_{max} = 0.5$ - Yu & Girimaji ($S_s = S_{max}\\sin{(\\omega t)}$)')

#mpl.subplot(1,2,1)
#DNS:
mpl.semilogy (k_x_DNS_w_med_norm,k_y_DNS_w_med_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.semilogy (t_st_norm_dict['w_med'],k_st_norm_dict['w_med'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.semilogy (t_real_norm_dict['w_med'],k_real_norm_dict['w_med'],'--',color = 'red',label='Realizable $k-\\epsilon
#Lumley k-eps:
mpl.semilogy (t_lum_norm_dict['w_med'],k_lum_norm_dict['w_med'],'--',color = 'green',label='Lumley $k-\\epsilon$')
#RSMS:
mpl.semilogy (t_rsms_norm_dict['w_med'],k_rsms_norm_dict['w_med'],color = 'blue',label='Standard RSM')
#RSMR:
mpl.semilogy (t_rsmr_norm_dict['w_med'],k_rsmr_norm_dict['w_med'],color = 'red',label='Realizable RSM')
#RSML:
mpl.semilogy (t_rsml_norm_dict['w_med'],k_rsml_norm_dict['w_med'],color = 'green',label='Lumley RSM')

#Formating(7,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_{max}*t$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,50)
#mpl.ylim (0,1)
mpl.xticks (np.linspace(0,50,6))
#mpl.yticks(np.linspace(0,1,3))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['w_med'],[x/Sh_w_med for x in S_lum_dims_dict['w_med']],'--',color = 'green',label=' $\
# #Sh:
# 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',lab

# #Formating(7,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_{max}*t$")
# mpl.xlim (0,50)
# mpl.xticks (np.linspace(0,50,6))
# mpl.ylim (-3,3)
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_med.png',dpi = plot_DPI)

##############################w_high NORMALIZED PLOTS:############################
mpl.figure(8)
mpl.suptitle('$\\omega/S_{max} = 1.0$ - Yu & Girimaji ($S_s = S_{max}\\sin{(\\omega t)}$)')

#mpl.subplot(1,2,1)
#DNS:
mpl.semilogy (k_x_DNS_w_high_norm,k_y_DNS_w_high_norm,'s',color='white',markeredgecolor='black',label='DNS')
#Standard k-eps:
mpl.semilogy (t_st_norm_dict['w_high'],k_st_norm_dict['w_high'],'--',color = 'blue',label='Standard $k-\\epsilon$')
#Realizable k-eps:
mpl.semilogy (t_real_norm_dict['w_high'],k_real_norm_dict['w_high'],'--',color = 'red',label='Realizable $k-\\epsil
#Lumley k-eps:
mpl.semilogy (t_lum_norm_dict['w_high'],k_lum_norm_dict['w_high'],'--',color = 'green',label='Lumley $k-\\epsilon$'
#RSMS:
mpl.semilogy (t_rsms_norm_dict['w_high'],k_rsms_norm_dict['w_high'],color = 'blue',label='Standard RSM')
#RSMR:
```

```
mpl.semilogy (t_rsmr_norm_dict['w_high'],k_rsmr_norm_dict['w_high'],color = 'red',label='Realizable RSM')
#RSML:
mpl.semilogy (t_rsml_norm_dict['w_high'],k_rsml_norm_dict['w_high'],color = 'green',label='Lumley RSM')

#Formating(8,1):
mpl.legend(loc='upper left',ncol=1)
mpl.xlabel("$S_{max}*t$")
mpl.ylabel("$k/k_0$")
mpl.xlim (0,50)
#mpl.ylim (0,1)
mpl.xticks (np.linspace(0,50,6))
#mpl.yticks(np.linspace(0,1,3))

#Uncomment to plot S as a function of time:
# ax = mpl.subplot(1,2,2)
# #Lumley k-eps:
# mpl.plot (t_lum_norm_dict['w_high'],[x/Sh_w_high for x in S_lum_dims_dict['w_high']],'--',color = 'green',label='
# #Sh:
# 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

# #Formating(8,2):
# mpl.legend(loc='upper right',ncol=1)
# mpl.xlabel("$S_{max}*t$")
# mpl.xlim (0,50)
# mpl.xticks (np.linspace(0,50,6))
# mpl.ylim (-3,3)
# ax.yaxis.tick_right()
# ax.yaxis.set_label_position("right")

fig = mpl.gcf()
fig.set_size_inches(plot_L, plot_W)
mpl.savefig('/home/dtobin/Documents/NASA Research/Report/w_high.png',dpi = plot_DPI)

#mpl.close("all")
##################################b11/b22 Test Plots##############################
mpl.figure(9)
mpl.title("$b_{11}$")
mpl.plot (t_rsms_norm_dict['w_med'][0:999],b11_rsms_dims_dict['w_med'],label = "$\\omega / S_{max} = 0.5$")
mpl.plot (t_rsms_norm_dict['w_high'][0:999],b11_rsms_dims_dict['w_high'],label = "$\\omega / S_{max} = 1.0$")
mpl.legend(loc='upper right',ncol=1)
mpl.xlabel("$S_{max}*t$")
mpl.ylabel("$b_{11}$")
mpl.xlim (0,50)
mpl.xticks (np.linspace(0,50,6))

mpl.figure(10)
mpl.title("$b_{22}$")
mpl.plot (t_rsms_norm_dict['w_med'][0:999],b22_rsms_dims_dict['w_med'],label = "$\\omega / S_{max} = 0.5$")
mpl.plot (t_rsms_norm_dict['w_high'][0:999],b22_rsms_dims_dict['w_high'],label = "$\\omega / S_{max} = 1.0$")
mpl.legend(loc='upper right',ncol=1)
mpl.xlabel("$S_{max}*t$")
mpl.ylabel("$b_{22}$")
mpl.xlim (0,50)
mpl.xticks (np.linspace(0,50,6))

#####Show Plots:#####
mpl.show(block=False)
```