ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
ΤΜΗΜΑ ΧΗΜΕΙΑΣ

ΤΙΤΛΟΣ ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

ΕΡΓΑΣΤΗΡΙΟ ΠΕΡΙΒΑΛΛΟΝΤΙΚΩΝ ΚΑΙ ΧΗΜΙΚΩΝ ΔΙΕΡΓΑΣΙΩΝ

ΜΕΤΑΠΤΥΧΙΑΚΟ ΔΙΠΛΩΜΑ ΕΙΔΙΚΕΥΣΗΣ

Προσαρμογή ενός συστήματος αφομοίωσης ατμοσφαιρικών δεδομένων για την μελέτη των εκπομπών μεθανίου στην Ελλάδα.

Ευαγγέλου Ιωάννα

Υπεύθυνη Καθηγήτρια: Κανακίδου Μαρία

ΗΡΑΚΛΕΙΟ 2022

UNIVERSITY OF CRETE
DEPARTMENT OF CHEMISTRY

TITLE OF POSTGRADUATE PROGRAMME

ENVIRONMENTAL CHEMICAL PROCESSES LABORATORY

Master Thesis

Adaptation of an atmospheric data assimilation system for the study of methane emissions in Greece.

Ioanna Evangelou

Master Thesis Supervisor: Maria Kanakidou

HERAKLION 2022

Εξεταστική Επιτροπή

Κανακίδου Μαρία

Καθηγήτρια (Επιβλέπουσα)

Κοσιώρης Γεώργιος

Καθηγητής

Μιχαλόπουλος Νικόλαος

Καθηγητής

ACKNOWLEDGMENTS

Words cannot express my gratitude to my professor and chair of my examinee committee Prof. Dr. Kanakidou for her invaluable patience, supervision and feedback. I also could not have undertaken this work without my defense committee, Prof. Dr Kosioris and Prof. Dr. Michalopoulos, who generously provided knowledge and expertise. Additionally, this endeavor would not have been possible without the support of my PhD supervisor and friend, Nikos Gialesakis, who helped, consulted and encouraged me in every step of this journey. I am also grateful to Dr. Nikos Daskalakis for the late-night feedback sessions and the moral support and all the members of ECPL group for their help. A special thanks also to Dr. Reum from DLR for giving us the WRF-GHG-CTDAS code as well as Prof. Dr. Vrekoussis and Dr. Schneising from IUP, University of Bremen for providing us with the TROPOMI/WFMD product. Lastly, I would be remiss in not mentioning my friends and family, especially my parents, and my brothers. Their belief in me has kept my spirits and motivation high during this process.

ΒΙΟΓΡΑΦΙΚΟ ΣΗΜΕΙΩΜΑ

ΙΩΑΝΝΑ ΕΥΑΓΓΕΛΟΥ

Ημερομηνία γέννησης: 18/09/1998, διεύθυνση: Αναγεννήσεως 42, Ηράκλειο Κρήτης, Ελλάδα, 71305, τηλ: 2810319518, κιν:(+30)6949846181, email:joanevangelou9881@gmail.com

ΕΚΠΑΙΔΕΥΣΗ

09/2016 – 06/2020: Πανεπιστήμιο Κρήτης, Τμήμα Φυσικής

•Πτυχίο: Επιστήμη της Φυσικής (ημερομηνία αποφοίτησης: Ιούλιος 2020)

•Βαθμός: 8,24/10, Διπλωματική εργασία: «Στατιστική ανάλυση και αξιολόγηση του αιολικού δυναμικού της Ανατολικής Κρήτης με χρήση πειραματικών και αριθμητικών δεδομένων», Επιβλέπων Καθηγητής: Γεώργιος Κοσιώρης, Τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών, UOC

09/2020 – μέχρι τώρα: Πανεπιστήμιο Κρήτης, Τμήμα Χημείας

•Μεταπτυχιακός τίτλος: Περιβαλλοντική Επιστήμη και Μηχανική

•Διατριβή: 'Αντίστροφη μοντελοποίηση για τη βελτίωση των εκτιμήσεων των ανθρωπογενών εκπομπών CH4 στην Ελλάδα με χρήση του WRF-GHG-CarbonTracker Data Assimilation Shell', Επιβλέπουσα Καθηγήτρια: Μαρία Κανακίδου, Τμήμα Χημείας, UOC

ΓΛΩΣΣΕΣ

•Μητρική Γλώσσα: Ελληνικά

•Αγγλική Γλώσσα (άπταιστα, επίπεδο C2, Michigan Proficiency, 2013)

•Γερμανική Γλώσσα (επίπεδο B2, Κρατικό Πιστοποιητικό Γλώσσας, 2013 / επίπεδο B1, Goethe-Institut, 2013)

ΕΡΕΥΝΗΤΙΚΑ ΕΡΓΑ

Διπλωματική εργασία: «Στατιστική ανάλυση και αξιολόγηση του αιολικού δυναμικού της Ανατολικής Κρήτης με τη χρήση πειραματικών και αριθμητικών δεδομένων». Έρευνα στο αιολικό δυναμικό με μοντέλο έρευνας καιρού Weather Research and Forecasting (WRF) (NCAR) για την περιοχή Σητείας Κρήτης και ανάλυση του ενεργειακού κέρδους από την εγκατάσταση ανεμογεννητριών.

ΕΡΕΥΝΗΤΙΚΑ ΑΡΘΡΑ

• Evangelou et al., 2021. Κατανομή μεθανίου στην Ελλάδα όπως προκύπτει από τα δεδομένα του Sentinel-5P TROPOMI, 17th International Conference on Environmental Science and Technology, 1-4 Sept, 2021

CURICULUM VITAE

IOANNA EVANGELOU

Date of birth:18/09/1998, address: Anagenniseos 42, Heraklion, Crete, Greece, 71305, tel:2810319518, mob:(+30)6949846181, email: joanevangelou9881@gmail.com

EDUCATION

09/2016 – 06/2020: University of Crete (UOC), Department of Physics

- Bachelor Degree: Science of Physics (graduation date: July 2020)
- Grade: 8.24/10 (243 ECTS), Diploma Thesis: 'Statistical analysis and evaluation of the wind potential of Eastern Crete with the use of experimental and numerical data', Supervisor Professor: Georgios Kosioris, Department of Mathematics and Applied Mathematics, UOC

09/2020 – until now: University of Crete (UOC), Department of Chemistry

- Master Degree: Environmental Science and Engineering
- Thesis: 'Inverse modelling for improving estimates of anthropogenic CH4 emissions over Greece using WRF-GHG-CarbonTracker Data Assimilation Shell', Supervisor Professor: Maria Kanakidou, Department of Chemistry, UOC

LANGUAGES

- Mother Language: Greek
- English Language (fluent, C2 level, Michigan Proficiency,2013)
- German Language (B2 level, State Certificate of Language,2013 / B1 level, Goethe- Institut, 2013)

RESEARCH PROJECTS

Bachelor Diploma thesis: 'Statistical analysis and evaluation of the wind potential of Eastern Crete with the use of experimental and numerical data'. Research on wind potential with weather research model Weather Research and Forecasting (WRF) (NCAR) for region Siteia, Crete and analysis of energy gain from installation of wind turbines.

PUBLICATIONS

- Evangelou et al., 2021. Methane distribution over Greece as derived from Sentinel-5P TROPOMI data, 17th International Conference on Environmental Science and Technology, 1-4 Sept, 2021

ΠΕΡΙΛΗΨΗ

Η υπερθέρμανση του πλανήτη που προκαλείται από τα αέρια του θερμοκηπίου είναι ένα θέμα άκρως ανησυχητικό τα τελευταία χρόνια. Το μεθάνιο (CH_4), που είναι το δεύτερο πιο σημαντικό αέριο θερμοκηπίου μετά το διοξείδιο του άνθρακα (CO_2), αυξάνεται με ταχείς ρυθμούς, κυρίως λόγω των ανθρώπινων δραστηριοτήτων. Στην παρούσα μελέτη, στοχεύουμε να βελτιστοποιήσουμε τις εκτιμήσεις των ανθρωπογενών εκπομπών CH_4 στην Κεντρική-Ανατολική Μεσόγειο και στην Ελλάδα, όπως αναφέρεται από την υπηρεσία Copernicus Atmosphere Monitoring Service, χρησιμοποιώντας αντίστροφη μοντελοποίηση.

Για το σκοπό αυτό, εφαρμόζεται και χρησιμοποιείται το μοντέλο Weather Research and Forecasting - Greenhouse Gas σε συνδυασμό με το CarbonTracker Data Assimilation Shell, που ενσωματώνει δορυφορικές παρατηρήσεις από το προϊόν TROPOMI/WFMD και επιτόπιες μετρήσεις από το σταθμό στο Φινοκαλιά Λασιθίου. Επιλέχθηκαν δύο εμφωλευμένες περιοχές μελέτης, η Κεντρική-Ανατολική Μεσόγειος και η Ελλάδα και βελτιστοποιούνται πέντε διαφορετικές γεωγραφικές περιοχές σε αυτήν την περιοχή μελέτης. Χρησιμοποιούνται 50 μέλη συνόλου με χρονικό βήμα βελτιστοποίησης μίας εβδομάδας και παράθυρο αφομοίωσης 5 εβδομάδων. Η περίοδος αντιστροφής ορίζεται στις πρώτες δέκα εβδομάδες του 2019.

Ερευνήσαμε δύο διαφορετικές περιπτώσεις, μία με σφάλμα μοντέλου μεταφοράς 10 ppb και 5 ppb για τα δορυφορικά δεδομένα και τις επίγιες μετρήσεις, αντίστοιχα και μία με δέκα φορές μεγαλύτερο σφάλμα. Σε όλες τις εβδομάδες που αντιστρέφονται για τις δύο προσομοιώσεις, προβλέπονται σχεδόν μόνο αρνητικές ροές. Αυτό αποδίδεται σε υψηλές αρχικές και οριακές συνθήκες σε σύγκριση με τις παρατηρήσεις που επιλέχθηκαν για το σύστημά μας. Επιπλέον, προβλέπονται διαφορετικοί αριθμοί περιοχών με θετικές εκπομπές από κάθε προσομοίωση, υποδεικνύοντας τη μεγάλη επίδραση του του σφάλματος του μοντέλου μεταφοράς στα αποτελέσματα εκτίμησης εκπομπών.

Λέξεις κλειδιά: μεθάνιο, εκπομπές, WRF, CTDAS, αντίστροφη μοντελοποίηση, σφάλμα μοντέλου μεταφοράς

ABSTRACT

Global warming induced by greenhouse gases has been an issue of outmost concern in the recent years. Methane (CH_4), which is the second most important greenhouse gas after carbon dioxide (CO_2), increases rapidly, mainly due to human activities. In the present study, we aim to optimize the estimations of CH_4 anthropogenic emissions over Central-Eastern Mediterranean and over Greece as reported by Copernicus Atmosphere Monitoring Service, using inverse modeling.

For this purpose, Weather Research and Forecasting - Greenhouse Gas model coupled with CarbonTracker Data Assimilation Shell, integrating satellite observations from TROPOMI/WFMD product and in-situ measurements from Finokalia station is implemented and used. Two one-way nested domains are selected, one over Central-Eastern Mediterranean and one over Greece and five different geographical regions in this study area are optimized. 50 ensemble members are used in the inversion with an optimization time step of one week and an assimilation window of 5 weeks. The inversion period is set to the first ten weeks of 2019.

We investigated two different setups, one with transport model error of 10 ppb and 5 ppb for satellite data and in situ measurements, respectively and one with ten times larger error. In all of the weeks inverted for the two simulations, almost only negative fluxes are predicted. This is attributed to high initial and boundary conditions compared to the observations that were selected to work as input in our system. Furthermore, different number of positive emission regions in the study area are predicted by each inversion setup indicating the large influence of transport model error magnitude in the emissions estimation results.

Keywords: Methane, emissions, WRF, CTDAS, inverse modeling, transport model error

CONTENTS

Page

Field Code Changed

Field Code Changed

ABBREVIATIONS

CH_4: Methane

CO_2: Carbon Dioxide

CO: Carbon Monoxide

CTDAS: CarbonTracker Data Assimilation Shell

EnSRF: Ensemble Square Root Filter

GHG: GreenHouse Gas

TM5: Transport Model 5

TROPOMI: TROPOspheric Monitoring Instrument

TROPOMI/WFMD: TROPOspheric Monitoring Instrument / Weighting Function Modified Differential Optical Absorption Spectroscopy

WRF: Weather Research and Forecasting model

WRF-GHG: Weather Research and Forecasting – Greenhouse Gas model

1. Earth

1.1 Climate and Greenhouse Effect

The term "climate" refers to the weather's average behavior over a long time period. It's not always easy to figure out what the typical time period is for defining the climate. Too short spans of time are insufficient to balance year-to-year variations, while too long periods can encompass numerous periods of climate change. Climate change has a characteristic time period ranging from decades to centuries. The average duration for defining the climate is 30 years. The global average annual surface temperature is the most commonly used variable to characterize climate, however other factors such as rainfall frequency and amount can also be considered. Changes in the average values of these factors, as well as their variability, are part of climate change. The balance controls surface temperature between solar energy inflow and the planetary heat emission in space.

Solar radiation is the primary source of energy for the Earth system. As a black body with an active temperature of T_S = 5800 K, Sun emits radiation. The black body's associated energy flow is given by the Stefan-Boltzmann law $F = \sigma T_S^4$, where σ is the Stefan-Boltzmann constant, equal to 5.67×10^{-8} W m^{-2} K^{-4} and F the flux through a surface perpendicular to the incoming radiation. Solar radiation has a peak at 0.5 μm and extends to all wavelengths. The Earth disk (surface perpendicular to the incoming radiation) interrupts the passage of solar energy by 1365 W m^{-2}. This amount is known as the solar constant and is represented by the letter S. As a result, the average solar flux received by Earth's disk is S / 4 = 341 W m^{-2}. Clouds and the Earth's surface reflect a fraction of the incoming solar energy back into space (around 30%). This is known as the planetary albedo (α=0.3). The Earth-atmosphere system absorbs the rest of the energy. The black body radiation emitted by the Earth at an active temperature T_E compensates for this energy input. The relationship between solar heating and ground cooling at steady state is:

$$(1-\alpha) \times S/4 = \sigma T_E^4$$

T_E = 255 K (-18 °C) is the average active temperature of Earth obtained from this equation. The wavelengths of terrestrial emission correspond to the infrared (IR), with a peak at 10 μm.

On average, roughly 341 W m^{-2} of solar energy enters the Earth's atmosphere each year, with 30% of the energy being reflected back into space, leaving about 235 W m^{-2} to be absorbed by the Earth/Atmosphere system (Fig. 1). Clouds account for around two-thirds of the 107 W m^{-2} reflected back into space, the surface for about one-eighth, and Rayleigh atmospheric scattering for the rest. Although the surface-atmosphere system's total absorbed and transmitted energy is balanced at 235 W m^{-2}, process fluxes and energy transfer within the system, between the surface outflux of Earth's radiation through the top of the Atmosphere and the Atmosphere, and within the Atmosphere itself, modify this balance and Earth's temperature from -18 ^{o}C (255 K) to +15 ^{o}C (288 K). In fact, certain gases, aerosols, and clouds in the atmosphere absorb around 70 percent of the 235 W m^{-2} of the outgoing longwave Earth's radiation. Indeed, infrared terrestrial radiation released from the surface is trapped by some gases, the so-called greenhouse gases. Therefore, the actual Earth's temperature is 33 K higher than the average active temperature. This is the natural greenhouse effect. Greenhouse gases are molecules that absorb infrared light and thereby lower the amount of terrestrial radiation that escapes into space warming the Earth/Atmosphere system. According to quantum physics, vibrational transitions are only allowed if they affect the molecule's bipolar moment. Greenhouse gases are any molecules with an asymmetric charge distribution (CO, chlorofluorocarbons) or the ability to obtain charge distribution through tension or bending (CO_2, CH_4, N_2O, H_2O, O_3).

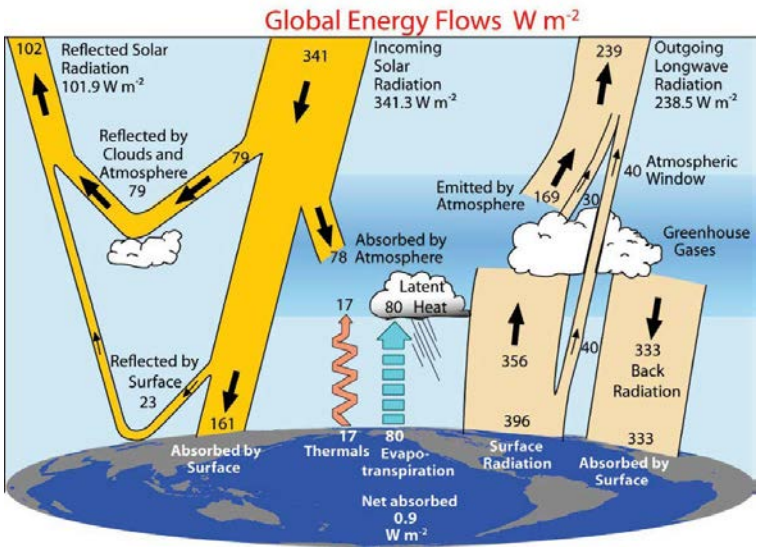


Figure 1. Global annual average energy balance of the Earth for the period 2000-2004. The units are W m^{-2}. (Brasseur & Jacob, 2017)

A change in the planet's energy balance at the top of the atmosphere that has the potential to modify global temperature, such as that resulting from a change in the quantity of sunlight that strikes the Earth or a change in the abundance of CO_2 in the atmosphere, is referred to as climate forcing. It has been observed that greenhouse gas (GHG) concentrations have risen during the last century. The resulting net trapping of infrared radiation in the Earth – Atmosphere system increases as GHG concentrations rise, contributing to the human greenhouse effect. The amount of rise in each GHG's concentration, the absorption spectrum of each, and potential interactions with other atmospheric components, all influence the ensuing climate change (Brasseur & Jacob, 2017).

1.2 Methane

1.2.1. Atmospheric importance of methane

Methane (CH_4) is a hydride of group-14 of periodic table, the simplest alkane. Naturally occurring methane is found both below ground and below the seabed and is formed by both geological and biological processes. When CH_4 reaches the surface and the atmosphere, it is known as atmospheric methane. Methane is second in importance to CO_2 among greenhouse gases with significant anthropogenic sources. Typically, in a time horizon of 100 years, CH_4 will be 28 times more efficient per mass as a greenhouse gas than CO_2 (Bruhwiler et al., 2014). In the atmosphere, it is gradually oxidized producing CO_2 and moisture (H_2O), which are also greenhouse gases.

An ongoing growth in global CH_4 is observed since the industrial revolution. In the pre-industrial period methane concentration was about 722 ppb_v, then increased and stabilized around the year 2000 to approximately 1775 ppb_v, resuming globally an increase in 2007 with the current global average around 1910 ppb_v (Fig. 2) (https://gml.noaa.gov/ccgg/trends_ch4/). As a greenhouse gas, CH_4 contributes around 20% of the total radiative forcing from all the long-lived globally mixed greenhouse gases (0.5 ± 0.05 W m^{-2}) (Zhou et al., 2018). Therefore, the significance of CH_4 and its potential to be a in global climate warming is beyond controversy.

Field Code Changed

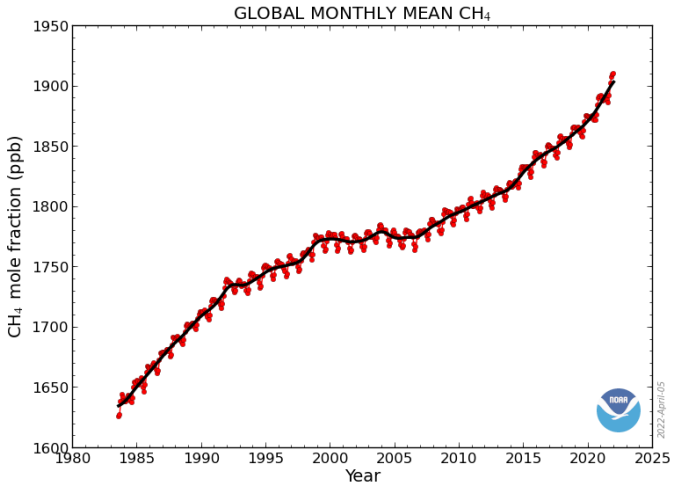


Figure 2. Global Methane trend since 1983 (source: https://gml.noaa.gov/ccgg/trends_ch4/)

Field Code Changed

As the governments became aware of the ongoing climate crisis, they signed one of the most important agreements for environment protection, the Paris Agreement in 2015, promising to confine global warming by the mid of 21st century to less than 2 ^{o} C or preferably, to less than 1.5 ^{o} C compared to the pre-industrial average temperature (https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement). For sustainable development of the society and in order to fulfill the commitment of the Paris Agreement, it is necessary to limit anthropogenic climate change through targeted emission reductions, especially emissions of greenhouse gases, such as CO_2 and CH_4. While CO_2 is the major contributor to the climate change, CH_4 compared to CO_2 has a short lifetime (about 10 years for methane, several decades to centuries for CO_2), therefore its atmospheric concentrations will respond faster than CO_2 to emission changes.

Field Code Changed

1.2.2. Methane cycle

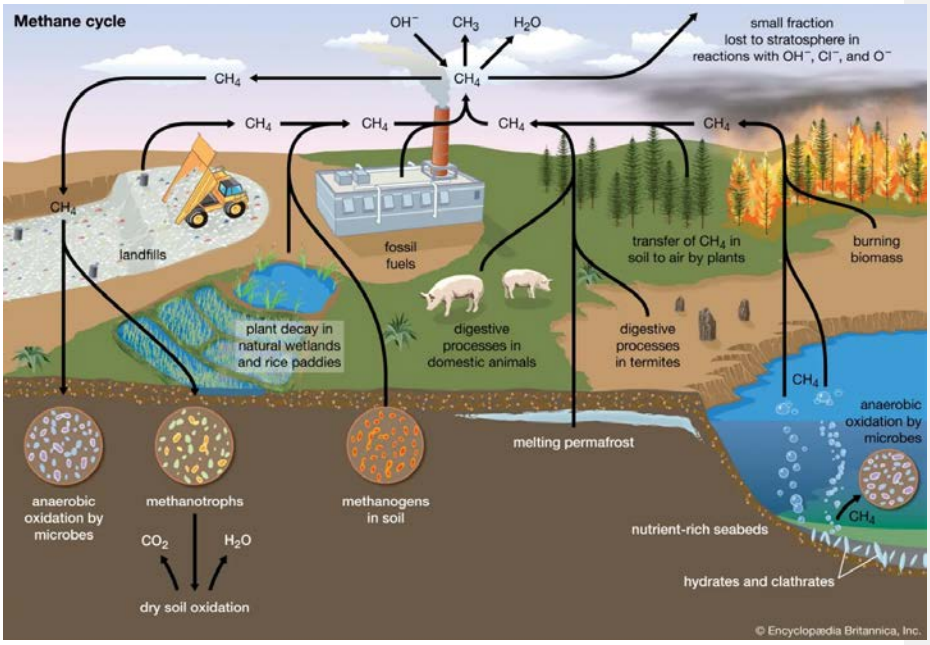


Figure 3. Methane cycle (source: Encyclopedia Britannica).

Methane sources are of both anthropogenic and biogenic origin, varying from fossil fuels and ruminants to wetlands and termites. Methane sinks in the troposphere are the reactions with hydroxyl radical, OH, chlorine radical, Cl both in the troposphere and stratosphere, and O^1D in the stratosphere as well as deposition to the surface (Fig. 3).

1.2.3. Major Methane sinks

O(1D), singlet oxygen, is produced by O_3 photolysis both in troposphere and stratosphere. In the troposphere, O^1D reacts with water vapor (H_2O) to produce hydroxyl radical, OH.

Globally, the main oxidation reaction of methane, is with the hydroxyl radical:

$$CH_4 + OH \rightarrow CH_3 + H_2O \qquad (reaction\ 1)$$

The methyl radical, CH_3, reacts instantaneously with O_2 to give methyl peroxy radical, CH_3O_2, which is also very reactive in the atmosphere (Müller et al., 2016) that will further react to form secondary products like formaldehyde, CO and ultimately CO_2.

$$CH_3 + O_2 + M \rightarrow CH_3O_2 + M \qquad (reaction\ 2)$$

so that CH_4 + OH reaction can therefore be written as:

$$CH_4 + OH \underset{O_2}{\rightarrow} CH_3O_2 + H_2O \qquad (reaction\ 3)$$

The rate coefficient for reaction 1 is $k_1 = 1.85 \times 10^{-12} \exp(-1690 / T)$ cm^3 $molecule^{-1}$ s^{-1} (Atkinson et al., 2006). The lifetime of CH_4 due to the reaction with OH is roughly 9 years at T = 273 K and [OH] = 10^6 molecules cm^{-3}. Despite its long lifetime, CH_4 has a major effect on background tropospheric chemistry because of its large concentration (Seinfeld et al., 1998).

Another sink is the atomic chlorine (Cl) radical. The free atomic chlorine radicals in the atmosphere react with methane, resulting in the formation of HCl and a methyl radical (CH_3) that will further react as above discussed.

A smaller sink of methane is the methanotrophic organisms in the soil. A group of bacteria leads to the oxidation of methane with nitrite as an oxidant in the absence of oxygen, causing the so-called anaerobic oxidation of methane. For simplicity, this sink is described in atmospheric numerical models as a dry deposition process.

1.2.4. Major Methane sources

In contrast to methane sinks, there is a wide variety of sources that significantly contribute to methane emissions. The following are the major methane sources we come across globally.

Biomass burning includes the incineration of living and dead materials in forests, savannas and agricultural waste and the incineration of fuel wood. Under ideal conditions of complete combustion, the combustion of the biomass material produces CO_2 and water vapor (H_20). Since complete combustion is not achieved under any biomass combustion conditions, other types of carbon, including carbon monoxide (CO), methane (CH_4), non-methane volatile organic compounds (NMVOC), and particulate carbon are produced (Fig. 4). The flaming phase is close to complete combustion, while the smoldering phase is close to incomplete combustion. Open air biomass combustion contributes between 20 to 60 Tg C yr^{-1} in the form of methane to the global atmosphere. This represents 5 to 15% of the world's annual methane emissions. Measurements show that biomass combustion is the overwhelming source of CH_4 in tropical Africa (Levine, 2000).

The contribution of biomass combustion to the total budget of methane or any other species depends on a variety of ecosystem and fire parameters, including the specific ecosystem burned, the mass consumed during combustion, the nature of the combustion (complete or incomplete), and knowledge of how emission factors (EF, amount of methane emitted per unit of burned material) vary depending on changing fire conditions in different ecosystems. The accuracy in the determination of all these parameters reflect in the accuracy of the estimate of biomass combustion emissions.

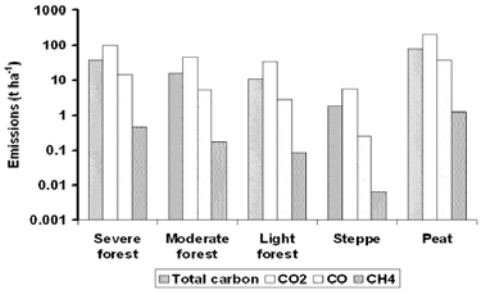


Figure 4. Estimates of carbon release and CO_2, CH_4, CO emissions from fires as a function of fuel type (steppe, forest, and peatland) and burn severity (severe, moderate, light) based on fraction of biomass (carbon) consumed. Uncertainties in these estimates are $\pm$ 50 % (Levine, 2000).

Hydrate gas is an ice-like substance formed when water and low molecular weight gases (CO_2, H_2S, CH_4 and higher order hydrocarbons) combine into a clathrate structure (Fig. 5). They are cage-like structures, with 1 m^3 of hydrated CH_4 resulting from trapping a maximum of 180 m^3 of methane as measured at standard temperature and pressure (STP). Hydrate gas, a naturally occurring and highly concentrated form of methane, traps significant carbon in the global system. It is widespread in the sediments of the marine continents and frost areas.

Hydrate gases are destabilized by increasing temperature or decreasing pressure, conditions that are rarely associated with the same climate. For elastic sediments, that are sediments that bend under load and recover when the load is removed, the pressure disturbance associated with sea level rise would be relatively instantaneous. Conversely, the impact of temperature changes on the tundra (hydrated permafrost) or seabed (marine or submarine hydrated permafrost) on deep-buried hydrates can be delayed by hundreds or thousands of years, depending on thickness and temperature of the supernatants, that is the upper layers of soil and liquid. This lag means that hydrate gas remains stable over centuries and in response to climate change can release significant methane gas on millennial scales (Ruppel & Kessler, 2017).

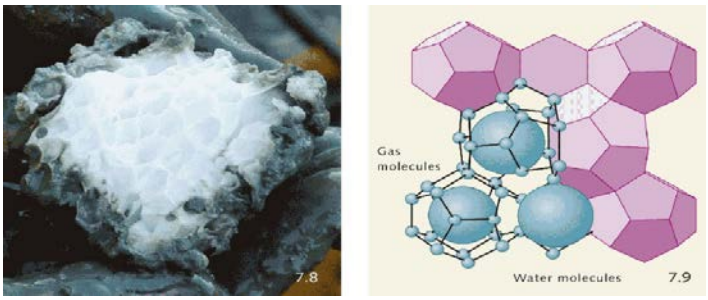


Figure 5. Gas hydrates structure (source: https://worldoceanreview.com/en/wor-1/energy/methane-hydrates/)

Field Code Changed

Landfills: CH_4 estimates for global waste disposal methane emissions range from 9 to 70 Tg yr^{-1}.

Methane is formed by methanogenic bacteria, either by decomposition of organic acids into CH_4 and CO_2, or by reducing CO_2 with hydrogen. Representative reactions are shown below:

$$CH_3COOH \rightarrow CH_4 + CO_2 \quad (reaction\ 4)$$

$$CO_2 + 4H_2 \rightarrow CH_4 + H_2O \quad (reaction\ 5)$$

The percentage of landfill carbon that is eventually converted to methane and carbon dioxide is not satisfactorily high. In the best case, 25% to 40% of waste carbon can be converted to biogas, that is methane produced by landfills and can be used as fuel. Field and laboratory studies suggest that maximum methane from landfill waste is approximately 0.06 to 0.09 m^3 per kg of dry waste depending on the moisture content and other variables such as organic load, storage capacity and nutrients. Landfill CH_4 emissions after 2012 show a rapid upward trend (Fig. 6), which reflects the growth trend of the urban population. As the rapid increase in the urban population can lead to rapid growth in the disposal of solid waste, if there is no significant implementation of landfill methane mitigation measures, a rapid increase in emissions should also be expected. (Bogner & Spokas, 1993)

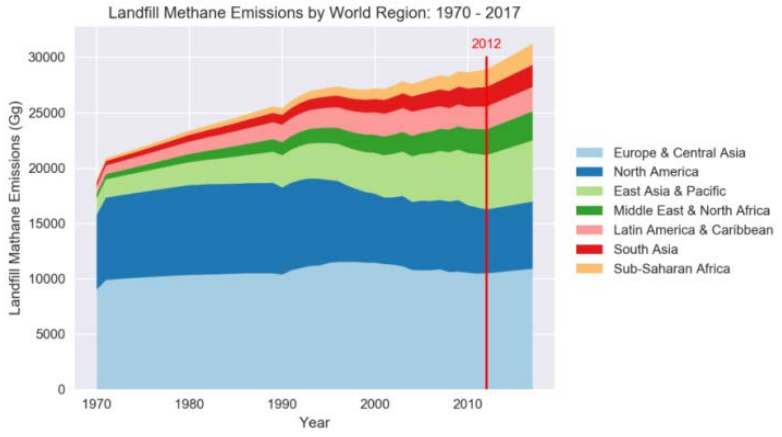


Figure 6. Landfill methane emissions by world region: 1970-2017 (Zhao et al., 2019)

Livestock is the largest anthropogenic source in the global methane budget (103 Tg CH_4 yr^{-1} mean over the period 2000–2009). Annual mean ruminant intestinal fermentation predominates in this source and represents emissions of 87–97 Tg CH_4 yr^{-1} during the period 2000–2009. Animal manure management has a smaller contribution. Cattle, buffalo, goats and sheep are the main types of animal ruminants that emit CH_4 and together account for 96% of the global source of intestinal fermentation.

The animal's microbiome is made up of bacteria, fungi, protozoa and archaea that turn grass into a source of energy for ruminants, but also produce methane as a by-product of fermentation. CH_4 is produced by a group of germs called methanogens (archaea). CH_4 is released into the atmosphere from the stomach through the animal's breath or during the storage of manure and pulp (Fig. 7) (Patra et al., 2017).

It is estimated that global $F_{CH4\text{-}ruminant}$ doubled from 48.5 ± 5.6 Tg CH_4 yr^{-1} in 1961 to 99.0 ± 11.7 Tg CH_4 yr^{-1} in 2012. The increase of emissions took place mainly in Latin America and the Caribbean, in East and Southeast Asia, in Sub-Saharan Africa, Near East and North Africa and South Asia. In contrast, from 1961 to 2012, $F_{CH4\text{-}ruminant}$ decreased in Europe and Russia by 31% and 54%, respectively, making 2012 emissions lower than those of 1961 in these two regions (Chang et al., 2019). Number of bovine animals in Europe can be shown in Fig. 8.

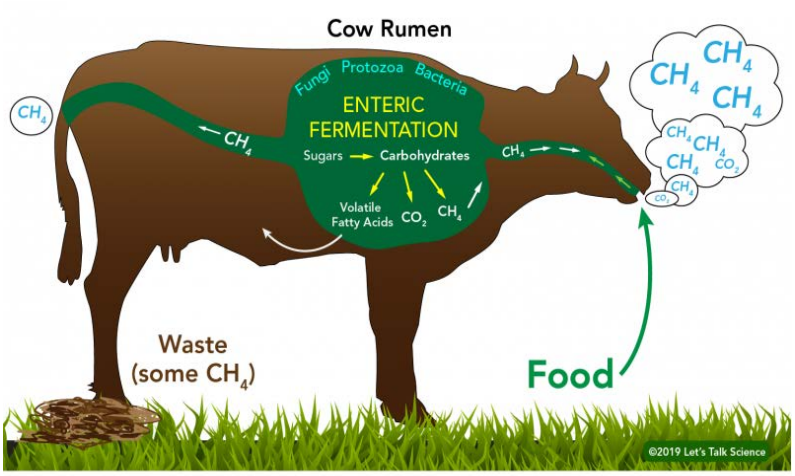


Figure 7. Enteric fermentation (source: https://letstalkscience.ca/)

Field Code Changed

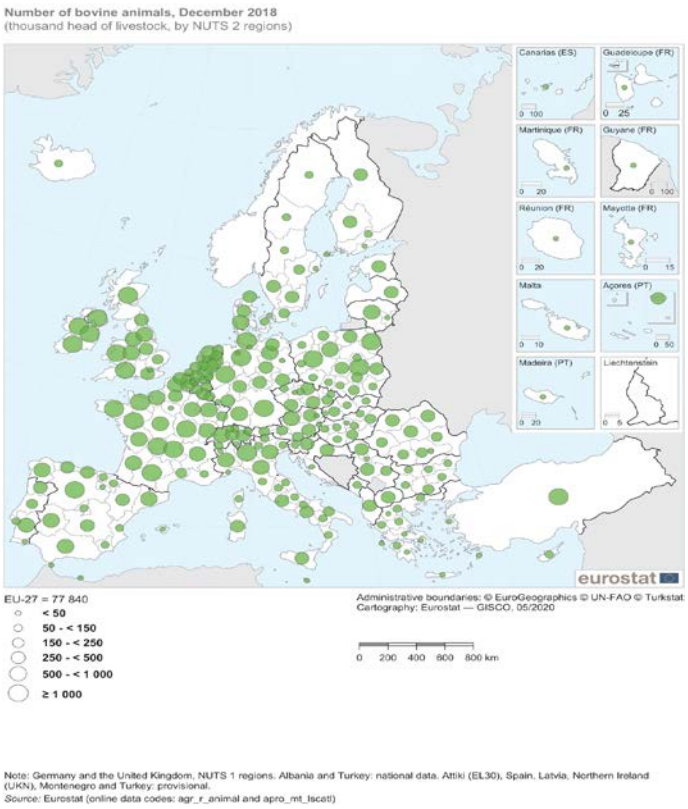


Figure 8. Number of bovine animals in Europe (source: Eurostat, https://ec.europa.eu/, last access: 3/2022).

Field Code Changed

Fuels: Methane can be emitted by **coal, oil and gas plants**. Natural gas is composed mainly of CH_4 and secondarily of ethane (C_2H_6) and propane (C_3H_8). Methane emissions are mainly due to leakage into the atmosphere during the various stages of natural gas exploitation and transportation, and from mines (Fig. 9). Fig. 10 shows the natural gas system in Greece.

The presence of C_2H_6 in the atmosphere can be an indicator for CH_4 emissions. Thermogenic and biogenic methane sources can be separated using the ethane-to-methane emission ratios. Although there are no relevant emissions of C_2H_6 during microbial methanogenesis, C_2H_6 is emitted together with CH_4 from thermal sources, i.e. mainly from the extraction of fossil fuels. The ethane-to-methane emission ratio (EMR) is greater than 1.0% for most methane thermogenic sources, while biogenic sources are characterized by EMR values below 0.1% (Hausmann et al., 2016).

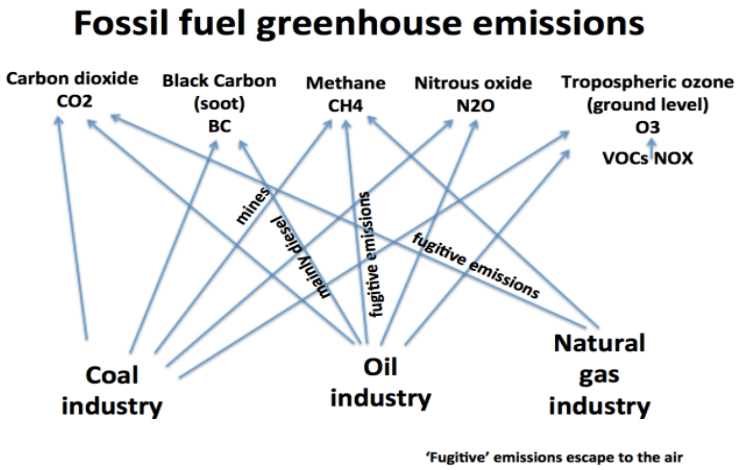


Figure 9. GHG and Black Carbon emissions by fossil fuel industry (source: https://tropicsu.org/lesson-plan-hydrocarbons-and-climate-change/)

Field Code Changed

Figure 10. Natural gas distribution system in Greece (source: https://www.desfa.gr/)

Field Code Changed

Rice fields: The main carriers responsible for methane's emissions from rice fields are methanogenic bacteria. Such microorganisms perform well under anaerobic conditions and are collecting organic carbon and converting it to methane. The starting materials for the involved reactions are the straws of

rice plants which represent the main inflow of organic matter (Fig.11). These materials usually accumulate during the rainy season or flood periods, decompose and become the main source of methanogenic substrates.

During the rainy season the emissions are higher, while at the same time there are reductions in rice production. On the other hand, during the dry season, methane emissions are lower. At the same time, rice yields are low during the dry season and high during the rainy season. This is explained by the fact that in the wet season, the resulting photosynthesis produces carbon, but the lack of flowers and blossoms makes this carbon unavailable for grain production and therefore low rice production. This amount of carbon that is not used for seed production enters the soil as rotten roots and as the leaf litter falls, it serves as a raw material for methanogens, leading to high CH_4 emissions in the wet season. In the dry season scenario, much of the carbon is used for sowing and active rice production, resulting in lower CH_4 emissions and higher rice production.

Globally, most of CH_4 emissions from rice fields occur in and around the tropics, subtropics and parts of the temperate north. Southeast Asia contributes 90% to the global rice emissions. Africa and South America add 3.5% and 4.7% to the global rice methane budget respectively. The percentage of CH_4 emissions from rice fields is increasing at a rate of 0.7-1.1% per year. This corresponds to 10-70% of man-made methane (Sanchis et al., 2012).

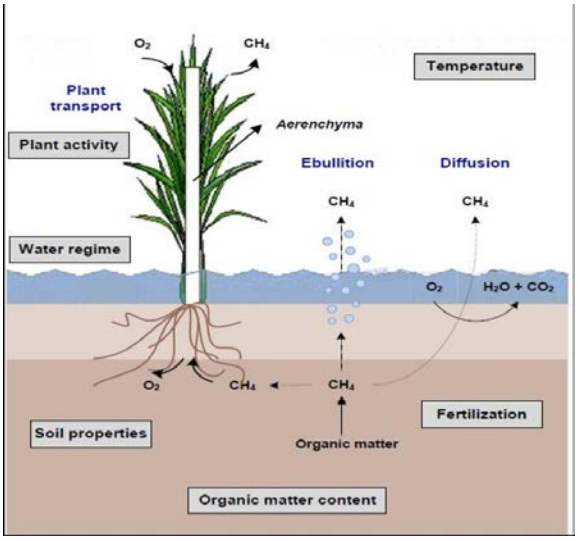


Figure 11. Methane emissions from rice fields (Sanchis et al., 2012).

Wetlands: Anaerobic CH_4 is also released by wetlands into the atmosphere after being affected by a combination of processes involving methanotrophic bacteria in the soil through plant transport structures known as "aerenchyma", by ebullition or through soil pores. Differences between current and prehistoric emissions are due to changes in wetland area and various variables, including nitrogen deposition, sedimentation, temperature, land use change and land cover. The largest change in wetland emissions from prehistoric industry occurred in the northern temperate zone (−79%) with smaller changes in high latitudes (+ 9%) (Fig. 12). In the northern regions, the increase in CH_4 emissions corresponds to the increase in wetlands and air temperature, while in the tropics the decrease in wetland area and the large fluctuation of rainfall are responsible for the reduced CH_4 emissions. The world's largest source of atmospheric CH_4 (1-2.37 × 10^5 Tg yr^{-1}) is acidic wetlands, such as peat and fences, which cover 3.5% of the earth's surface but store about 30% of the global carbon of terrestrial ecosystems. Acidic wetlands can be temporarily converted into atmospheric CH_4 sinks, or at least have the potential to consume atmospheric CH_4, which is currently underestimated and poorly understood at the microbial level (Kolb & Horn, 2012) (Poulter et al., 2017).

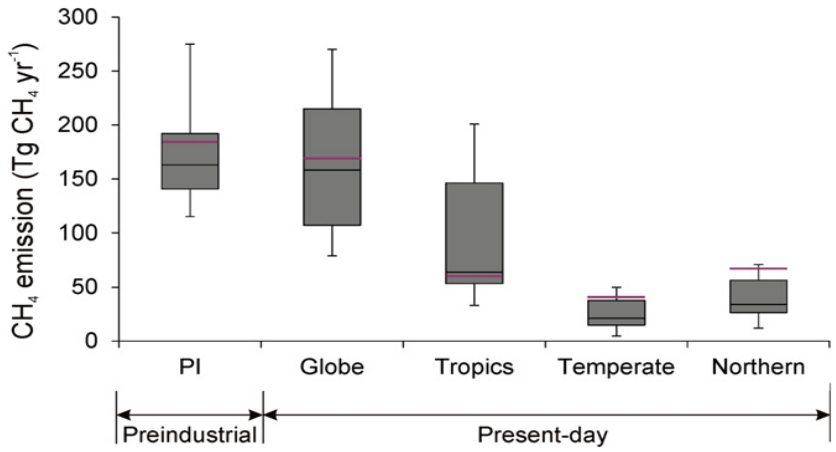


Figure 12. Wetland methane emissions estimates for the preindustrial (PI) and present-day conditions for the entire Globe. Preindustrial emissions are mean values from 1850 to 1869 (PI simulation) and present-day emissions are simulated mean values from 1993 to 2004. The box shows the interquartile range, the whiskers show the maximum and minimum, and black lines gives the median methane emissions. The purple line gives the results from Paudel et al. (2016).

Each of these sources and sinks contributes to methane abundances in a different extent. Fig.13 depicts the mean global methane budget for the period 2000 to 2009. Methane emissions from human activities have surpassed natural emissions since the 1980s (Salawitch et al., 2017). As above discussed, human activities that produce methane include energy production from natural gas, coal and oil, decomposition in landfills, ruminant farming and rice cultivation. Wetlands are the main natural source of methane; while biomass burning emissions have a significant human contribution. Despite the importance of methane as a greenhouse gas in the Earth's atmosphere, there are still great uncertainties about the location and intensity of emission sources. The two major difficulties in reducing uncertainty stem from the wide variety of geographically overlapping diffuse CH_4 sources and the destruction of CH_4 by the very short-lived OH radical.

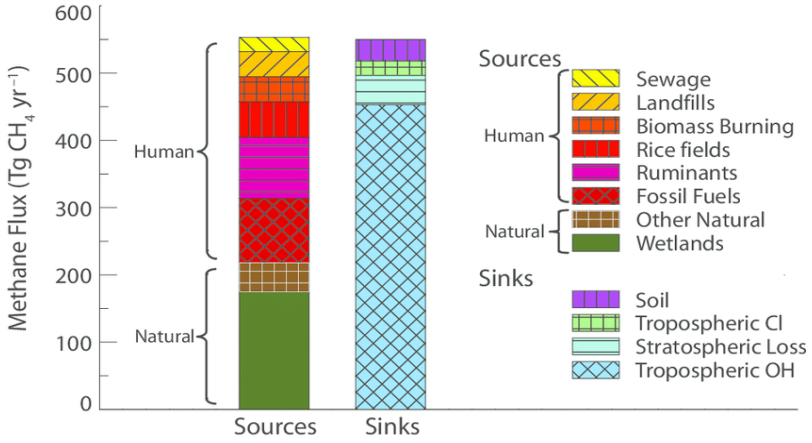


Figure 13. Global methane budget (2000-2009) from (Salawitch et al., 2017).

1.2.5. CH_4 sources in Greece

Specifically, in Greece waste is the most significant source of anthropogenic methane emissions, accounting for about 45% of total methane emissions in 2019 (without LULUCF). Since 1990, methane emissions from waste have been reduced by 1.35%, owing primarily to solid waste disposal on land and wastewater treatment. Agriculture's methane emissions declined by 9.1 % in 2019 compared to 1990 levels. Agriculture accounted for 44.6% of total methane emissions in 2019, with enteric fermentation being the primary source category in the sector. Energy-related methane emissions (mostly fugitive emissions from coal mining and the production, processing, and distribution of liquid fuels and natural gas) account for nearly 10% of overall methane emissions (Fig. 14) (Ministry of Environment and Energy, 2021).

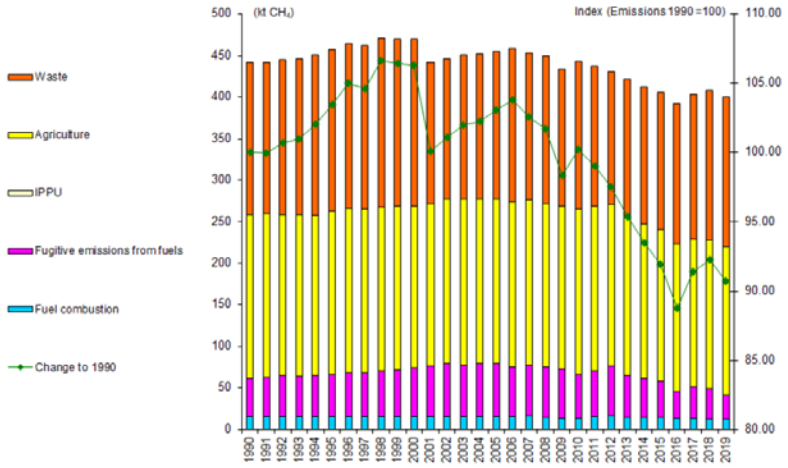


Figure 14. Anthropogenic CH_4 emissions in Greece by major sectors for the period 1990 – 2019. Left axis provides emissions in kt CH_4 yr^{-1}. IPPU is the Industrial Processes and Product Use sector. Right axis provides percent change compared to 1990 emissions (Ministry of Environment and Energy, 2021).

1.3 Aim of the study

In the present study, we aim to improve the estimations of the sum of methane total anthropogenic emissions over Central-Eastern Mediterranean as well as over Greece as provided by Copernicus Atmosphere Monitoring Service version 4.2 emission dataset, with the aid of inverse modeling.

For this purpose, we use the Weather Research and Forecasting-Greenhouse Gas numerical model (Beck et al., 2011) coupled with CarbonTracker Data Assimilation Shell (Van Der Laan-Luijkx et al., 2017), the optimization algorithm of which is the Ensemble Square Root Kalman Filter.

Regarding the observations against which the emissions are optimized, satellite observations from TROPOspheric Monitoring Instrument WFMD product from Institute of Environmental Physics (IUP), University of Bremen (Schneising et al., 2019) and in-situ measurements from Finokalia station (htpp://finokalia.chemistry.uoc.gr), Lasithi from Environmental Chemical Processes Laboratory, University of Crete are integrated into the model system.

2. Methodology

2.1. Inverse modeling

In the present study, we attempted to improve methane anthropogenic emissions through data assimilation. Data assimilation is a time-stepping technique in which we optimize the variables driving a physical system, using observations of that system. In atmospheric chemistry, data assimilation is commonly referred as a type of problem in which we aim to optimize a gridded time-dependent 3-D model field of atmospheric concentrations or emissions based on measurements of these concentrations or associated factors.

The variables we want to optimize are referred to as state variables, and they are assembled into a state vector x. We construct the observations into an observation vector y in the same way. The forward model of the physical system, model F, describes our understanding of the link between x and y:

$$y = F(x, p) + \varepsilon_0 \quad (1)$$

where, p is a parameter vector that includes all model variables that we don't want to optimize as part of the inversion, and ε_0 is an observational error vector that includes contributions from measurements, the forward model, and model parameters errors. The forward model predicts the effect (y) as a function of the cause (x) using equations that describe the system's physics. We can quantify the cause (x) from observations of the effect by inverting the model (y). The solution is a best estimate of x, which is called the ideal estimate, posterior estimate, or retrieval. Other limitations on the value of x may help to reduce the error on the optimal estimate due to the uncertainty in deriving x from y. This is called prior information. The prior estimate x_A, which represents our best guess of x before the observations are made and has an error of ε_A, is a typical constraint used in assimilation. The best estimate is calculated based on the error statistics of ε_0 and $x_{A,}$ (Brasseur & Jacob, 2017).

2.2. CarbonTracker Data Assimilation Shell

CarbonTracker is a data assimilation system for CO_2 that predicts global carbon sources and sinks. It was created at the National Oceanic and Atmospheric Administration's (NOAA) Earth System Research Laboratory (ESRL) between 2005 and 2007 (Peters et al., 2005). Following that,

development and application were split into two branches: (1) CarbonTracker (NOAA/ESRL) and (2) CarbonTracker Europe (CTE). Here we refer to the CTE version.

Using atmospheric observations of CO_2 mole fractions, the CarbonTracker data assimilation system for CO_2 estimates the carbon exchange between the atmosphere, land biosphere, and seas. TM5 transport model which connects surface fluxes to atmospheric CO_2 mole fractions, is a major component of CarbonTracker (Van Der Laan-Luijkx et al., 2017). In 2005, the existing TM5 CO_2 model version was also coupled with Carbon-Tracker, which required only a little amount of new code to use as a CO_2 ensemble Kalman smoother. New CarbonTracker requirements evolved over time, requiring the handling of new and more sophisticated data structures and work flows, which were difficult to implement in Fortran and not necessarily consistent with the continued development of TM5. This resulted in the CarbonTracker Data Assimilation Shell, a new object-oriented Python programming language implementation (CTDAS). It is built in a modular manner, allowing for the addition of new observation types, changes to the structure of the underlying state vector, and even the replacement of the transport model (e.g. WRF-GHG) or optimization method (e.g. four-dimensional variational) with only minor additional code within a single module (Van Der Laan-Luijkx et al., 2017).

In our study, CTDAS is coupled with the WRF-GHG transport model (Beck et al., 2011). Using observations of atmospheric CH_4 mole fractions, the CarbonTracker data assimilation system for CH_4 estimates methane emission fluxes. CarbonTracker is a *fixed-lag ensemble Kalman smoother application* based in the Bayesian approach (Peters et al., 2005). In the following, we explain how Carbon Tracker works with regard to methane.

The cost function (J) that represents the accuracy with which the system is solved, is used to optimize the surface CH_4 fluxes:

$$J(x) = \left(y^0 - H(x)\right)^T \boldsymbol{R}^{-1}\left(y^0 - H(x)\right) + (x - x^b)^T \boldsymbol{P}^{-1}(x - x^b) \quad (2)$$

where y_o are the atmospheric CH_4 mole fraction observations, with their error covariance R (model-data-mismatch error). H is the observation operator, which is an atmospheric transport model. H connects the observations y_o to the scalars that modify the surface CH_4 fluxes, which are included in the state vector x, with their error covariance R. The background state vector x_b with error covariance P contains prior information about the emission fluxes. Superscript T corresponds to the transpose of the matrix.

The state vector x and its covariance P that minimizes J can be shown to be:

$$x_t{}^a = x_t{}^b + \boldsymbol{K}(y_t{}^0 - H(x_t{}^b)) \quad (3)$$

$$\boldsymbol{P}_t{}^a = (\boldsymbol{I} - \boldsymbol{KH})\boldsymbol{P}_t{}^{\boldsymbol{b}} \quad (4)$$

in which t is a subscript for time, superscript b refers to background quantities and a to analyzed ones, **H** is the linear(ized) matrix form of the observation operator H, I is the identity matrix and K is the *Kalman gain matrix* defined as:

$$\boldsymbol{K} = \left(\boldsymbol{P}_t{}^{\boldsymbol{b}}\boldsymbol{H}^{\boldsymbol{T}}\right)(\boldsymbol{H}\boldsymbol{P}_t{}^{\boldsymbol{b}}\boldsymbol{H}^{\boldsymbol{T}} + \boldsymbol{R})^{-\mathbf{1}} \quad (5)$$

Kalman gain is the weight assigned to the measurements and current-state estimate. When the error covariance matrix of the observation is very large compared to the state vector (R>>P), then K is nearly 0, meaning you trust the model more than the measurements. When R<<P, then K is nearly 1 meaning you trust the measurement more than the model. In other words, with a large gain, the filter gives measurements more weight, and so conforms to them more quickly. The filter follows more closely to the model predictions when the gain is low. A high gain close to one will provide a jumpier estimated prediction, whereas a low gain close to zero will smooth out noise but reduce responsiveness.

In an ensemble Kalman filter, the information in the covariance matrix P is represented in fewer dimensions N by an ensemble of state vectors x_i composed of a mean state, and deviations from the mean state:

$$x_i = \bar{x} + x'_i \quad (6)$$

The deviations x'_i are created such that the normalized ensemble of deviations defines the columns of a matrix **X:**

$$\boldsymbol{X} = \frac{1}{\sqrt{N-1}}\,(x'_1, x'_2, \ldots, x'_N)^{\boldsymbol{T}} = \frac{1}{\sqrt{N-1}}\,(x_1 - \bar{x}, x_2 - \bar{x}, \ldots, x_N - \bar{x}) \quad (7)$$

which is the square root of the covariance matrix

$$\boldsymbol{P} = \boldsymbol{X}\boldsymbol{X}^{\boldsymbol{T}} \quad (8)$$

When N → ∞ this representation of P is exact, while in an ensemble Kalman filter with a finite number of members, P is approximated.

Whitaker & Hamill (2002) provided an efficient approach for calculating an optimized ensemble from a background ensemble with the correct covariance structure. This is the *ensemble square root filter* (EnSRF). The batch of observations relating to one filter time step are processed one at a time in the *sequential/serial* EnSRF algorithm, reducing the size of the Kalman gain matrix K in each sequential analysis step to a vector that has the size of the number of unknowns. The Kalman gain matrix is derived using the following approximations from the ensemble of state vectors and equation (5):

$$\boldsymbol{H}\boldsymbol{P}\boldsymbol{H}^{\boldsymbol{T}} \approx \frac{1}{\sqrt{N-1}}\left(H(x'_1), H(x'_2), \dots, H(x'_N)\right) \cdot \left(H(x'_1), H(x'_2), \dots, H(x'_N)\right)^T \quad (9)$$

$$\boldsymbol{P}\boldsymbol{H}^{\boldsymbol{T}} \approx \frac{1}{\sqrt{N-1}}(x'_1, x'_2, \dots, x'_N) \cdot \left(H(x'_1), H(x'_2), \dots, H(x'_N)\right)^T \quad (10)$$

where each entry N denotes one column of ensemble state vectors or ensemble modeled CH_4 values as in equation (7).

The Kalman gain matrix is used to update the mean state vector with equation (3), whereas the deviations from the mean state vector are updated independently using:

$$x'_i{}^a = x'_i{}^b - \tilde{\boldsymbol{k}} H(x'_i{}^b) \quad (11)$$

Where the vector $\tilde{\boldsymbol{k}}$ is related to the Kalman gain matrix **K** by a scalar quantity α calculated as:

$$\tilde{\boldsymbol{k}} = \boldsymbol{K} \cdot a \quad (12)$$

$$a = (1 + \sqrt{\frac{\boldsymbol{R}}{\boldsymbol{HP^bH^T+R}}})^{-1} \text{ (13)}$$

The analyzed mean and ensemble state from one observation will serve as the background state for the next until all observations are processed.

To reflect the additional information in the updated state vectors, we must also update the ensemble of sampled CH_4 concentrations $H(x_i{}'^b)$. Each modeled CH_4 concentration that corresponds to a yet-to-be-assimilated observation m (denoted $H(x_t)_m$ here) is updated using the equation:

$$H({x_t}^a)_m = H({x_t}^b)_m + \boldsymbol{H_m K}({y_t}^0 - H({x_t}^b)) \text{ (14)}$$

whereas the deviations are updated using:

$$H({x'_i}^a)_m = H({x'_i}^b)_m - \boldsymbol{H_m}\tilde{\boldsymbol{k}}({y_t}^0 - H({x'_i}^b)) \text{ (15)}$$

In the right-hand side of this equation the operator H_m has been substituted by its matrix equivalent $\mathbf{H_m}$.

The so-called dynamical model M plays a crucial role in data assimilation. Before fresh observations are provided to the system, this model predicts the evolution of the state vector through time and hence offers an initial guess of the state vector:

$$\boldsymbol{x}_{t+1}{}^b = M(\boldsymbol{x}_t{}^a) \text{ (16)}$$

In our case we use (16) to propagate the mean of the state, and prescribe its covariance structure at each new step drawing a new ensemble of N flux deviations from a specified background covariance to represent the Gaussian Probability Density Function around the flux.

CarbonTracker calculates scaling factors (λ_r) that multiply anthropogenic emissions. For each spatial region r and each time step (t) we want to better quantify the emissions, the total carbon fluxes F(x,y,t) are represented by:

$$F(x,y,t) = \lambda_r \cdot F_{anthro}(x,y,t) + F_{fire}(x,y,t) + F_{bio}(x,y,t) \quad (17)$$

The scaling vectors (λ_r) multiply F_{anthro}, which are pre-calculated space–time anthropogenic emissions obtained from Copernicus Atmosphere Monitoring Service (prior fluxes, see section 2.4.3). Fire emissions and biogenic fluxes are not optimized and we assume 100% certainty for them.

The obtained modeled mole fractions are compared to atmospheric data in CTDAS, and the differences are minimized by changing the flux scaling vectors (λ_r) resulting in optimum posterior fluxes. For each new time step t, the background scaling factors (λ^b) are chosen as the average of the optimal scaling factors (λ^a) from the two preceding time steps, plus the fixed prior value (λ^p) we select, as in the following equation by (Van Der Laan-Luijkx et al., 2017).

$$\lambda_t{}^b = \frac{\lambda_{t-2}{}^a + \lambda_{t-1}{}^a + \lambda^p}{3.0} \quad (18)$$

When inferring to fixed-lag assimilation window, we mean that instead of solving the Bayesian system in one large operation, smaller subsets of unknowns are optimized in a time stepping approach as in Bruhwiler et al. (2005). In the fixed lag ensemble square root Kalman filter used in CTDAS, the state vector contains flux estimates for multiple time steps t each corresponding for instance, to an one-week mean. This is indicated by the system's ''lag''. In other words, the relationship between the state vector x and observations y described by operator H spans several timesteps t.

A CTDAS cycle proceeds as follows (Fig. 15):

(1) We run the forward model from the background concentration fields in $CH_{4i}(x,y,z,t)$ to $CH_{4i}(x,y,z,t + 12)$ forced by the fluxes in xi(0,. . .,11) (A to B in the figure), and extract CH_4 mixing ratios at the observation times and locations. This allows us to construct an ensemble of modeled CH_4 at each site.

(2) Equations (3) and (11) are solved to give an analyzed ensemble of fluxes for each element of the state vector and each week.

(3) The ensemble of final fluxes in $x_i^a(12)$ will no longer be estimated in the next cycle and are therefore incorporated into $CH_{4i}(x,y,z,t + 1)$ by running the forward model one week forward starting from $CH_{4i}(x,y,z,t)$ forced with the final ensemble fluxes $x_i(12)$ (A to C in the Fig. 15).

(4) Each analyzed state vector becomes the background state vector for the next cycle. A new background mean flux is created to go into x(0) by propagation with model M (equation (16)).

(5) We draw a new ensemble of N flux deviations from the specified background covariance structure to represent the Gaussian Probability Density Function around the new mean flux x(0), and finally

(6) new observations y are read and the next cycle starts (Peters et al., 2005).

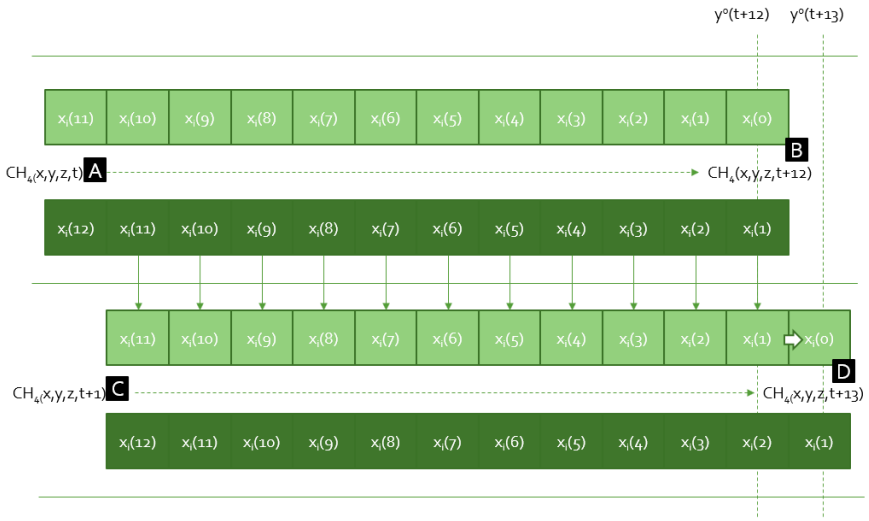


Figure 15. CTDAS cycles: 12 weeks of fluxes compose the state vector. Light shaded boxes denote the background fluxes, and dark shaded boxes denote posterior fluxes. Each box represents N ensemble members. The number in parentheses indicates how many times a week of fluxes has been estimated previously from past cycles, and the subscript i refers to an individual ensemble member.

The ensemble Kalman filter looks for correlations between random flux perturbations and simulated methane measurement variations. We could anticipate the entire ensemble to agree that increasing methane flux in one place results in higher simulated methane concentrations at a downwind site

nearby. However, because we utilize a random sample of a few ensemble members to approximate the flow covariance matrix, we occasionally see misleading correlations. Any link between the flux ensemble and the measurement could be fictitious.

For this reason, the CT2007 localization technique is used. For instance, if 150 ensemble members are used, the linear correlation coefficient between the 150 scaling factor deviations and the 150 observation deviations for each parameter/observation combination is calculated. The association between a parameter deviation and its modeled observational impact is kept, if it is statistically significant. Otherwise, due to the small ensemble's numerical estimate of the covariance matrix, the association is thought to be spurious noisy. Relationships that reach 95% significance in a Student's T-test with a two-tailed probability distribution are accepted in our case. (https://gml.noaa.gov/ccgg/carbontracker/CT2007/documentation_assim.html)

Field Code Changed

2.3. Weather Research and Forecast – Greenhouse Gas model

The forward model we utilize in the present study is the Weather Research and Forecasting – Greenhouse Gas model (WRF-GHG) reported in (Beck et al., 2011).

WRF contains two dynamical solvers, referred to as the advanced research WRF (ARW) and the nonhydrostatic mesoscale model (NMM). WRF ARW was used for the present study. The mesoscale model WRF (Skamarock et al., 2008) is a numerical weather prediction system that may be used for atmospheric research as well as operational forecasting at scales ranging from tens of meters to thousands of kilometers. First, WRF was combined with the Vegetation Photosynthesis and Respiration module to construct high-resolution regional simulations of atmospheric CH_4 passive tracer transport (WRF-VPRM) (Ahmadov et al., 2009). Then, WRF-VPRM was expanded to WRF-GHG (Beck et al., 2011), which can model the regional passive tracer transport for GHGs (CH_4, CO_2) and carbon monoxide (CO). In the last versions of WRF, WRF-GHG is integrated in WRF-Chem code (WRF model coupled with Chemistry) (Grell et al., 2005) as an individual chemistry option.

2.4 Model and Inversion Framework setup

2.4.1. Observations

The observations we utilize in the assimilation, in order to improve methane anthropogenic emissions, are satellite data as well as in situ measurements. Satellite data have exceptional temporal as well as horizontal resolution and a wide-region coverage in comparison with in-situ measurements. However, station observations are considered more accurate due to the procedure they are obtained.

Satellite data: TROPOMI/WFMD Methane column-averaged dry air mole fractions (XCH_4) v1.5 product is used as created by the Institute of Environmental Physics, University of Bremen (Schneising et al., 2019).

TROPOMI is an imaging spectrometer onboard of the Copernicus Sentinel-5 Precursor (S5P), a European satellite for atmosphere monitoring, launched on 13 October 2017 and planned for a mission of seven years. S5P is a sun-synchronous orbit satellite at 824 km altitude, with an Equator overpass time at 13:30 local time and a 16-days cycle. The swath of TROPOMI is about 2600 km and it operates with a horizontal resolution of 7×7 km^2 (5.6 × 7 km^2 from 6 August 2019).

To retrieve CH_4 from satellite observations, the Weighting Function Modified Differential Optical Absorption Spectroscopy (WFM-DOAS) algorithm that is a least-squares method was used by Schneising et al. (2019). By normalizing the vertical column amounts of CH_4 with the dry air column acquired from the European Centre for Medium-Range Weather Forecasts (ECMWF) analysis, the column-averaged dry air mole fractions of methane (denoted XCH_4) are generated.

To use TROPOMI/WFMD observations, modifications had to be made to the Python script for satellite observations reading since WRF-GHG-CTDAS initial setup is for handling O-CO2 satellite data (https://ocov2.jpl.nasa.gov/). The corresponding code is displayed in Appendix C. For comparison of observations to the model, the column averaging kernel as provided by the TROPOMI/WFMD product is applied to the model profiles using the formula:

Field Code Changed

$$X_{mod} = \sum_l ((X_{apr}{}^l + A_l(X_{mod}{}^l - X_{apr}{}^l))w_l \quad (20)$$

where l is the index of the vertical layer, A_l the averaging kernel, ${X_{apr}}^l$ the a-priori mole fraction and ${X_{mod}}^l$ the simulated mole fraction of layer l. w_l is the layer dependent pressure weight.

Due to the long lifetime of methane, it can be assumed that methane columnar values account for the background concentration of methane in the atmosphere (Appendix A).

In-situ observations: We used methane near surface measurements from the Finokalia atmospheric observatory (Fig. 16). The atmospheric measurement station of the University of Crete, a climate change observatory, established and operated by Environmental Chemical Processes Laboratory (E.C.P.L.) at Finokalia, Lasithi since 1993, holds the largest time series of atmospheric measurements of greenhouse gas concentrations throughout the Eastern Mediterranean. The station represents the Eastern Mediterranean atmosphere and has drawn significant scientific interest in the domains of atmospheric composition, air quality, and climate change. It also serves as the regional background station for Greece, reporting on air pollution levels to Greek and EU authorities. Methane measurements are performed discontinuously since 2002 (flask samples) and continuously by a PICARRO analyzer since June 2014 in collaboration with the Laboratory of Sciences of Climate and Environment (LSCE) in France. For the present study hourly CH_4 data are used.

Since point observation handling was not implemented in WRF-GHG-CTDAS, we developed the code displayed in Appendix C, based on the existing satellite data sampling scripts. For comparison of the model with the in-situ observations, the level of WRF-GHG which is closer to the station's altitude is selected. The model output of methane concentrations is in ppm and methane concentrations as reported by the station are given in ppb, thus the comparison is quite direct.

Figure 16. Finokalia (FKL) station (longitude: 25.670, latitude: 35.338, altitude: 250) (source: https://finokalia.chemistry.uoc.gr/gallery/station/).

Field Code Changed

2.4.2 Model Setup

This study uses version 4.3 of WRF coupled with Chemistry (WRF-Chem) (Grell et al., 2005). Methane is implemented as a passive tracer (chemistry option = 17 in the WRF-Chem code), thus not impacting meteorological variables and chemistry. Thus, the total mass of methane in the model domain depends only on the surface fluxes inside the domain and the boundary conditions.

There are no chemical processes in the WRF-GHG simulations for methane, despite the fact that oxidation by OH is the principal sink of methane in the atmosphere. This would not alter our results because the lifetime of methane is rather long compared to the simulation duration.

In our study we simulate the period 1/1/2019-11/3/2019. We use two nested domains for WRF-GHG (Fig. 17) on Lambert Conformal projection in order to simulated methane fluxes. The coarser domain (D01) has a horizontal grid distance of 36 km centering at 38.45 °N and 20.336 °W and it covers Central and Eastern Mediterranean. The domain has 33 vertical levels up to 50 hPa (about 20 km height). The finest domain (D02) is located over Greece with spatial resolution of 12 km, as shown in Fig. 17. The simulations were run with one-way nested mode.

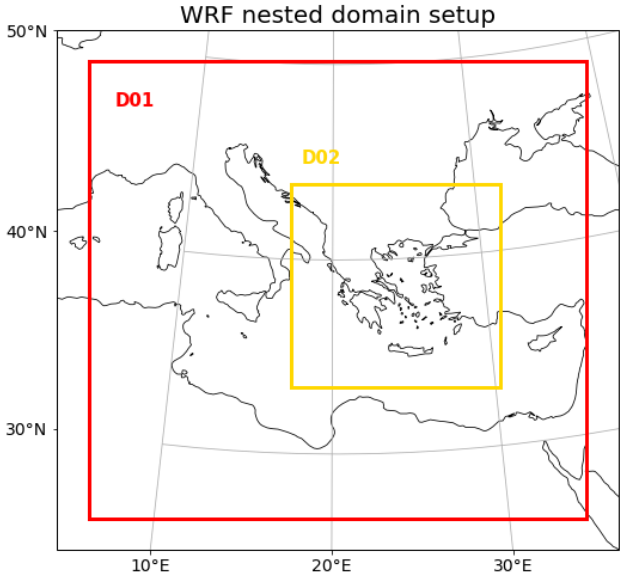


Figure 17. WRF-GHG domains used in our study.

We use ERA5 meteorology from European Centre for Medium Range Weather Forecasts (Hersbach et al., 2020) and improved orography from European Digital Elevation Model (EU-DEM) (https://land.copernicus.eu/imagery-in-situ/eu-dem/) in order to have better wind intensity and direction calculation, which can affect our emissions estimation. Initial and boundary chemical conditions for methane are obtained from the TM5 global chemistry transport model (Huijnen et al., 2010).

Field Code Changed

As for the parameterizations that are used in the present study, for the planetary boundary layer we selected the YSU scheme. The YSU scheme uses a 1st order closure to calculate the turbulent vertical fluxes within the planetary boundary layer. Kain-Fritsch (new Eta) scheme, a deep and shallow sub-grid scheme using a mass flux approach with downdrafts and CAPE removal time scale, is selected as cumulus scheme. For the representation of land surface processes, shortwave radiation and longwave radiation the default options are selected, which are respectively Noah Land-Surface Model, a unified NCEP/NCAR/AFWA scheme with soil temperature and moisture in four layers, fractional snow cover and frozen soil physics, Dudhia scheme, a simple downward integration allowing for efficient cloud and clear-sky absorption and scattering, and RRTM Rapid Radiative Transfer Model that accounts for multiple bands, trace gases, and microphysics species.

2.4.3 Apriori Emissions

Two different methane **emission databases** are used in this study. Emissions from biomass burning are taken from the Fire INventory from NCAR (FINN) version 2.4 and Copernicus Atmosphere Monitoring Service (CAMS) version 4.2 emission datasets are used for emissions by anthropogenic activities.

The Fire INventory from NCAR (FINN) model predicts worldwide emissions from open burning at high horizontal resolution of 1 km^2 and daily. FINN provides open burning emissions estimates for use in regional and global chemical transport models by combining satellite images of active fires and land cover with emission factors and estimated fuel loadings. The datasets used in the study are available at https://www.acom.ucar.edu/Data/fire/ .

Field Code Changed

The Copernicus Atmosphere Monitoring Service provides monthly gridded global emission inventories with spatial resolution of 0.1^ox0.1^o. These inventories describe anthropogenic emissions from fossil fuel use on land, ships, and aviation, as well as natural emissions from vegetation, soil, the ocean, and termites, using a combination of current data sets and new information. Anthropogenic emissions on land are further broken down into

different activity sectors (e.g., power generation, road traffic, industry). Because most inventory-based data sets are only accessible after several years, the CAMS emission inventories employ trends from the most recent available years to extend current data sets forward in time, providing timely input data for real-time forecast models. The anthropogenic inventories used in the present study are available at https://eccad3.sedoo.fr/ . Table 1 displays the average anthropogenic and fire emissions for the year 2019 as derived by CAMS and FINN inventories respectively for the two domains of our study.

Field Code Changed

Table 1. Average anthropogenic and fire emissions for 2019 for the two domains of the present study as derived by CAMS and FINN inventories respectively.

Emissions (mol km^{-2} hr^{-1})	*Domain 1*	*Domain 2*
Anthropogenic	17.26	17.90
Fire	0.0048	0.0056

The biogenic fluxes for methane (emissions from wetlands and termites and soil uptake) are calculated online in the WRF-GHG based on the work of Kaplan (2002), Sanderson (1996) and Ridgwell et al. (1999) respectively. Input fields of wetland fraction per grid cell and fast carbon pool are necessary for the calculation of the wetland emissions. We used the wetland map (Fig. 18) from the global dataset of Wetland Area and Dynamics for Methane Modeling (WAD2M) (Zhang et al., 2021) (https://essd.copernicus.org/articles/13/2001/2021/essd-13-2001-2021-discussion.html) and the fast carbon pool map (Fig. 19) from the Lund-Potsdam-Jena model (LPJ) (Sitch et al., 2003) as obtained by the Lawrence Livermore National Laboratory (https://esgf-node.llnl.gov/search/esgf-llnl/).

Field Code Changed

Field Code Changed

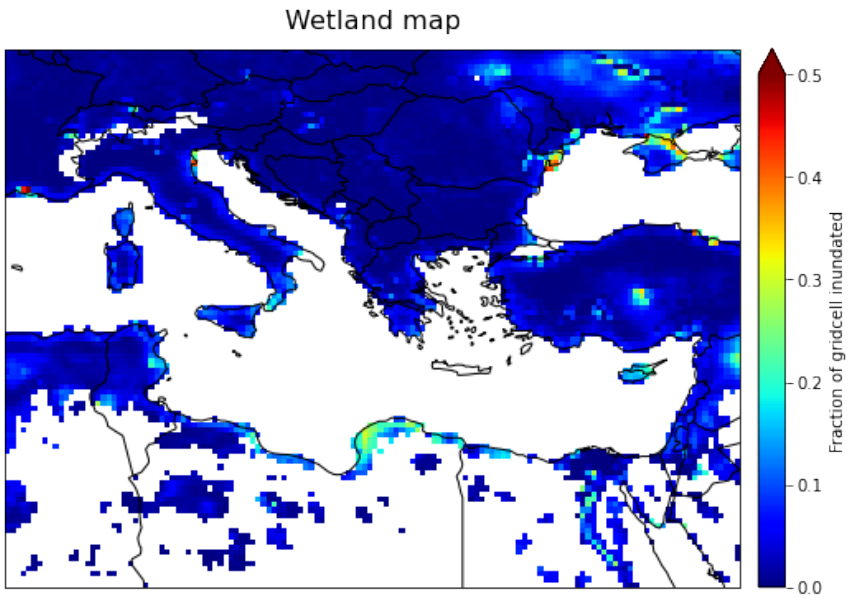


Figure 18. Inundated soil map for our coarser domain by WAD2M 0.25^{o} x 0.25^{o} product.

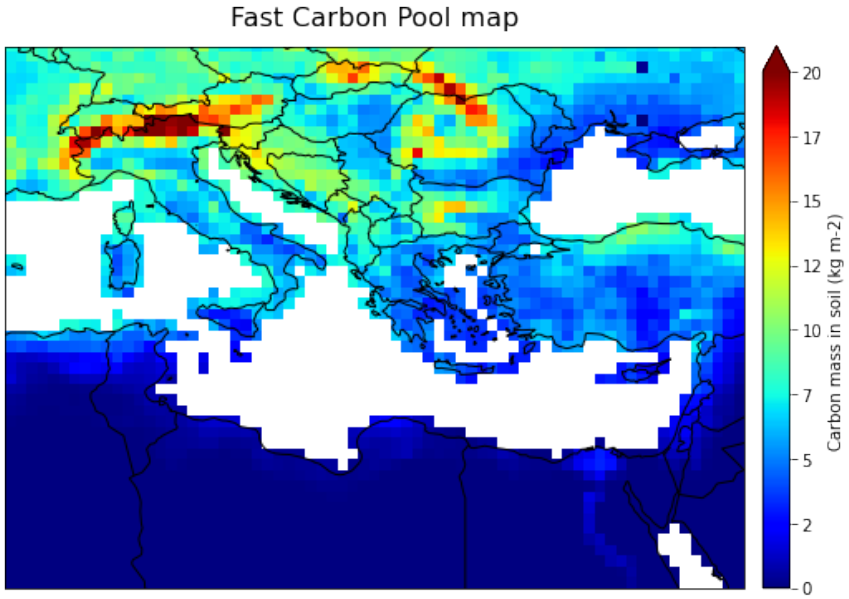


Figure 19. Fast carbon pool for our coarser domain by Lund-Potsdam-Jena 0.5^{o} x 0.5^{o} product.

2.4.4. CTDAS setup

The inversion system of CTDAS for WRF-GHG was downloaded from the Wageningen University and Research CTDAS repository (https://git.wur.nl/ctdas/CTDAS/-/tree/ctdas-wrf). We use a flux model in which we assume biosphere and fire fluxes fixed, and anthropogenic emissions with 100% uncertainty. The statevector consists of 5 parameters for mapping anthropogenic fluxes shown in Fig. 20. These will be referred to the next parts of the study as: Greece, North region, South region, East region and West region, with respect to the geographical position of Greece. Regarding Kalman filter setup, we use serial optimization algorithm, 50 ensemble members with an optimization time step of 7 days and an assimilation window (lag) of 5 weeks. The inversion is done for 10 CTDAS cycles, that is for period 1/1/2019-11/3/2019.

Field Code Changed

The model data mismatch R, that is the observational error in equation (1) and the matrix R in the equation (2), is set to the sum of squares of the observation product error and the transport model error. In other words, besides the measurement uncertainty(mdm), the combined uncertainty ($mdm_{combined}$) is calculated by adding an estimate for the model error as below:

$$mdm_{combined} = \sqrt{mdm^2 + transport\_error^2}$$

where mdm is the observational error, transport_error is the assumed error of WRF-GHG and of the concentrations mapping into the observations space and $mdm_{combined}$ is the overall uncertainty. In an inversion, the model-data mismatch covariance matrix (R) shows how well the optimized fluxes should be able to reproduce atmospheric observations, given errors in modeled transport, measurement, and gridded fluxes (Gourdji et al., 2018).

Satellite observations used are assigned an uncertainty as provided by the TROPOMI product and are selected only if they have a quality flag equal to zero, indicating only good CH_4 observations. For in situ data, we select mdm to be equal to the measurement standard deviation. The transport error is assumed to be 10 ppb and 5 ppb for satellite data and in-situ measurements, respectively. In a second run, we use 100 ppb and 50 ppb transport model error to investigate the difference it makes in the results. In the next sections we refer to these runs as 10 ppb and 5 ppb run, and 100 ppb and 50 ppb run. As for the observation rejection threshold, we set 99.7% probability threshold

(3σ) that observation agrees with the model prediction. In other words, if the inequality $|observation - H(x)| > 3 \times \sqrt{R}$ is true, the observation is considered as an outlier and it is not assimilated. Furthermore, we use 95% probability threshold that observation and state vector element correlate through CT2007 localization option.

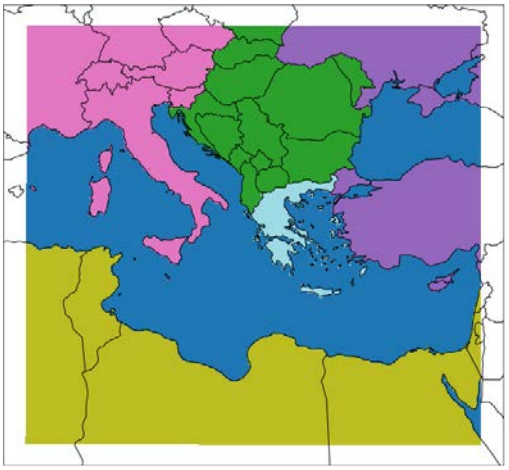


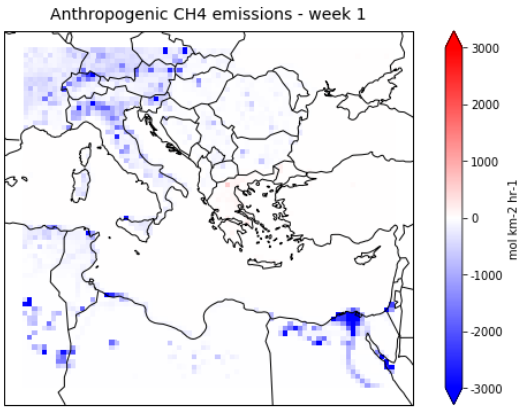
Figure 20. Statevector parameters regions used for our inversion setup: Greece (light blue), North region (green), South region (yellow), East region (purple) and West region (pink).

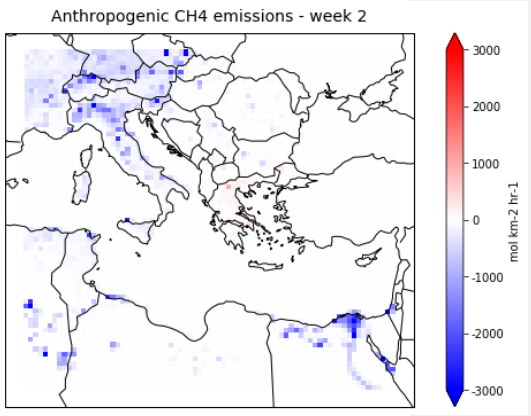
3. Results and Discussion

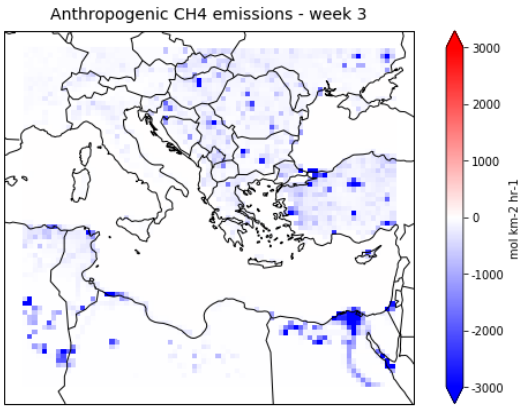
3.1 Inversion results

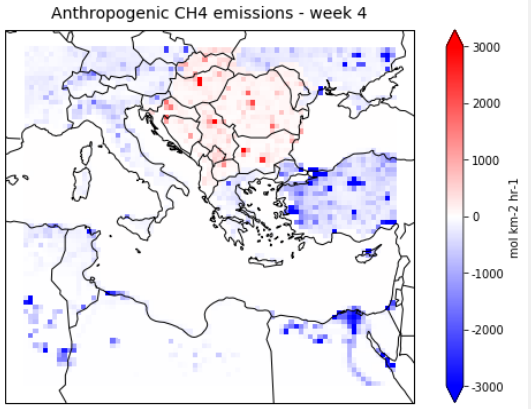
Two different runs were conducted with different transport model errors. In Fig. 21, the emission maps for one day of each week of our inversion period are depicted, which correspond to the 10 ppb (for satellite error) and 5 ppb (for in-situ error) run, hereafter called 10.5 run. In Fig. 22 the corresponding 50 ensemble statevector deviations for Greece are shown. Only the maps for the coarse domain are included here. The emission maps for the domain over Greece are available in the Appendix B.

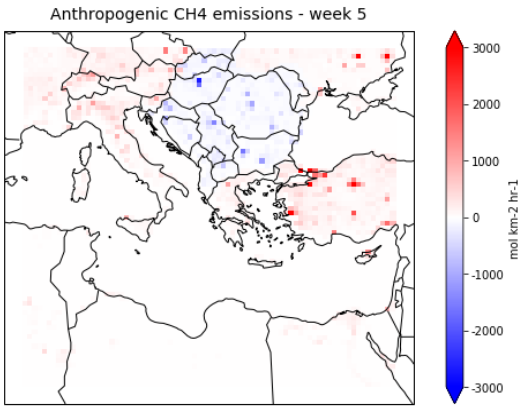
Regarding the 10.5 run, we observe that for the first 5 weeks, which also correspond to our system's lag, both negative and positive surface fluxes are predicted. After this period, the majority of the surface fluxes are negative. It is deduced that during the first 5 weeks, high TM5 initial and boundary conditions influence the results in large extent, forcing our system to significantly reduce the emissions in order to conform with observations. After 5 weeks, we assume that WRF-GHG has been stabilized and boundaries have a lower impact on the results, however still considerable. In addition to high boundary conditions, negative emissions could be attributed to the lack of chemistry in WRF-GHG and specifically of OH sink. The omission of chemistry is potentially leading to methane accumulation over our domain, which has a similar effect as high boundary conditions.

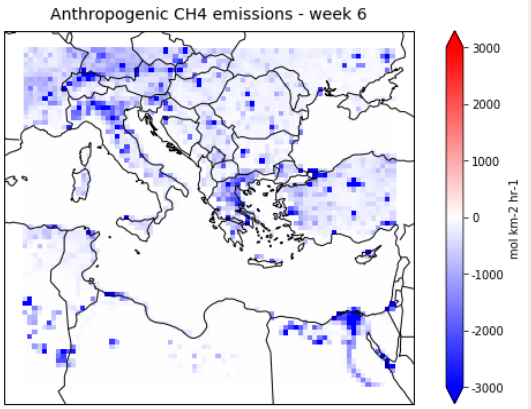

Anthropogenic CH4 emissions - week 1
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 2
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 3
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 4
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 5
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 6
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1

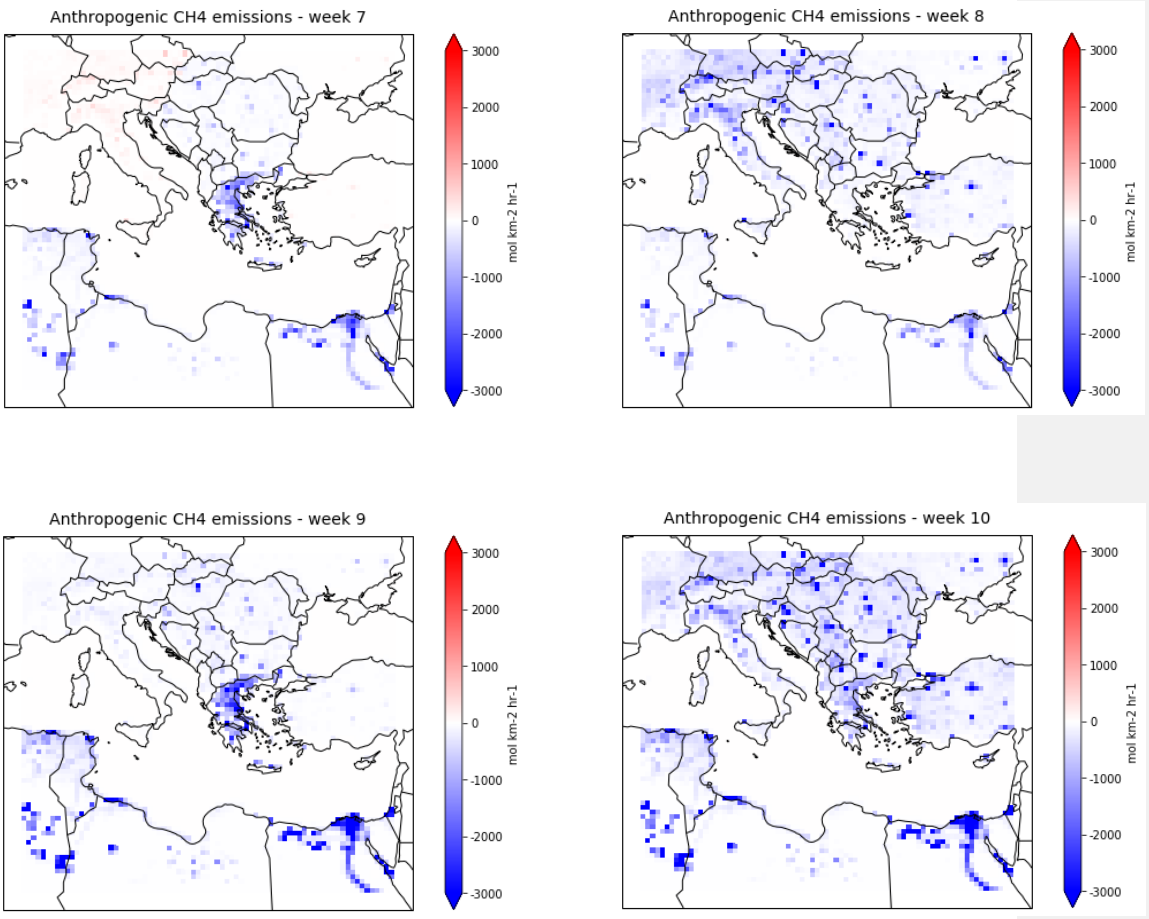
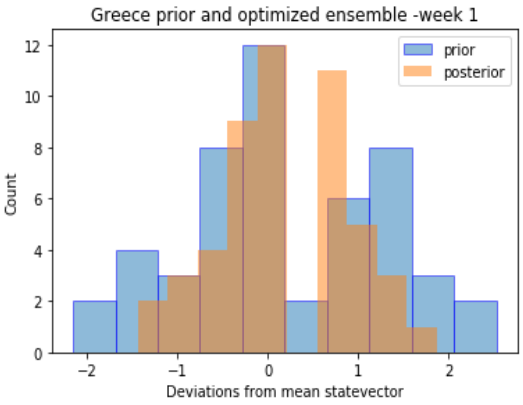
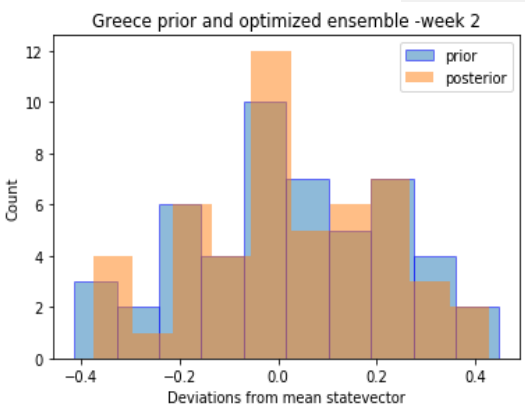
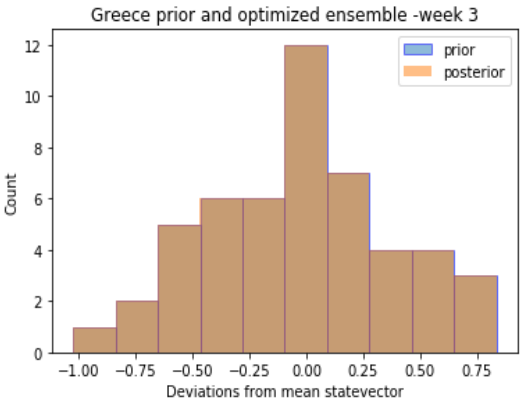


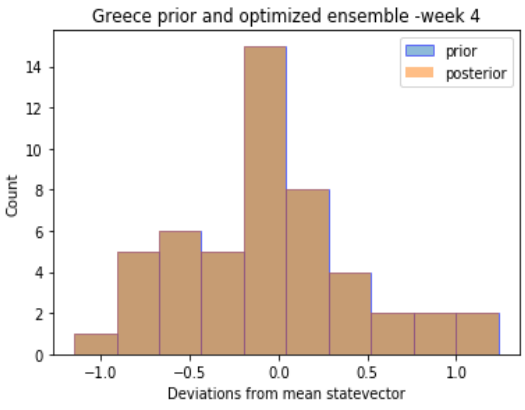
Figure 21. Optimized anthropogenic methane emissions over Eastern Mediterranean for the 10 weeks of the inversion period (10.5 run for the period 1/1/2019-11/3/2019).

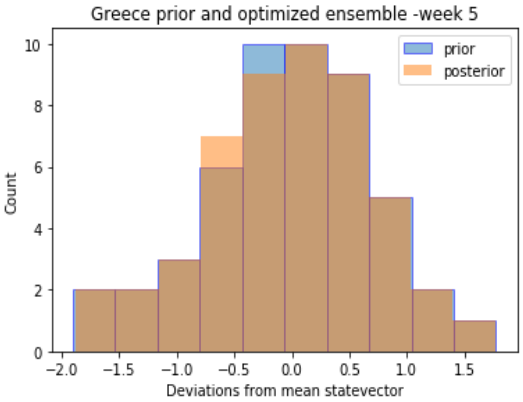
Fig. 22 shows the deviations around mean scaling vector, λ_i for Greece for the apriori and the posteriori ensembles. It can be seen that the deviations are decreased after the optimization, in other words, the prior error of λ is highly reduced which indicates that our system works properly in terms of the cost function minimization, reducing error in emissions. Optimization in the first cycles is more “abrupt” in comparison with the following ones, since λ in the next inversion cycles is closer to the “truth”; the assumption for λ in the first cycle is 1.

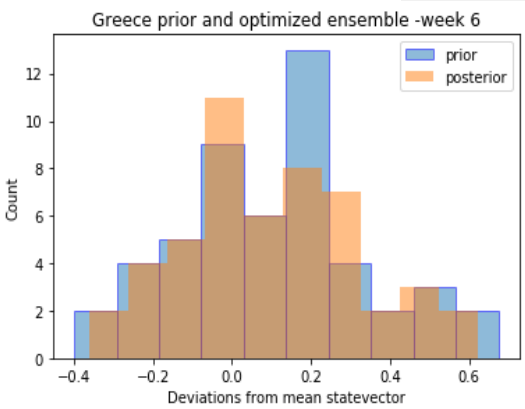

Greece prior and optimized ensemble -week 1
prior
posterior
Count
Deviations from mean statevector


Greece prior and optimized ensemble -week 2
prior
posterior
Count
Deviations from mean statevector


Greece prior and optimized ensemble -week 3
prior
posterior
Count
Deviations from mean statevector


Greece prior and optimized ensemble -week 4
prior
posterior
Count
Deviations from mean statevector


Greece prior and optimized ensemble -week 5
prior
posterior
Count
Deviations from mean statevector


Greece prior and optimized ensemble -week 6
prior
posterior
Count
Deviations from mean statevector

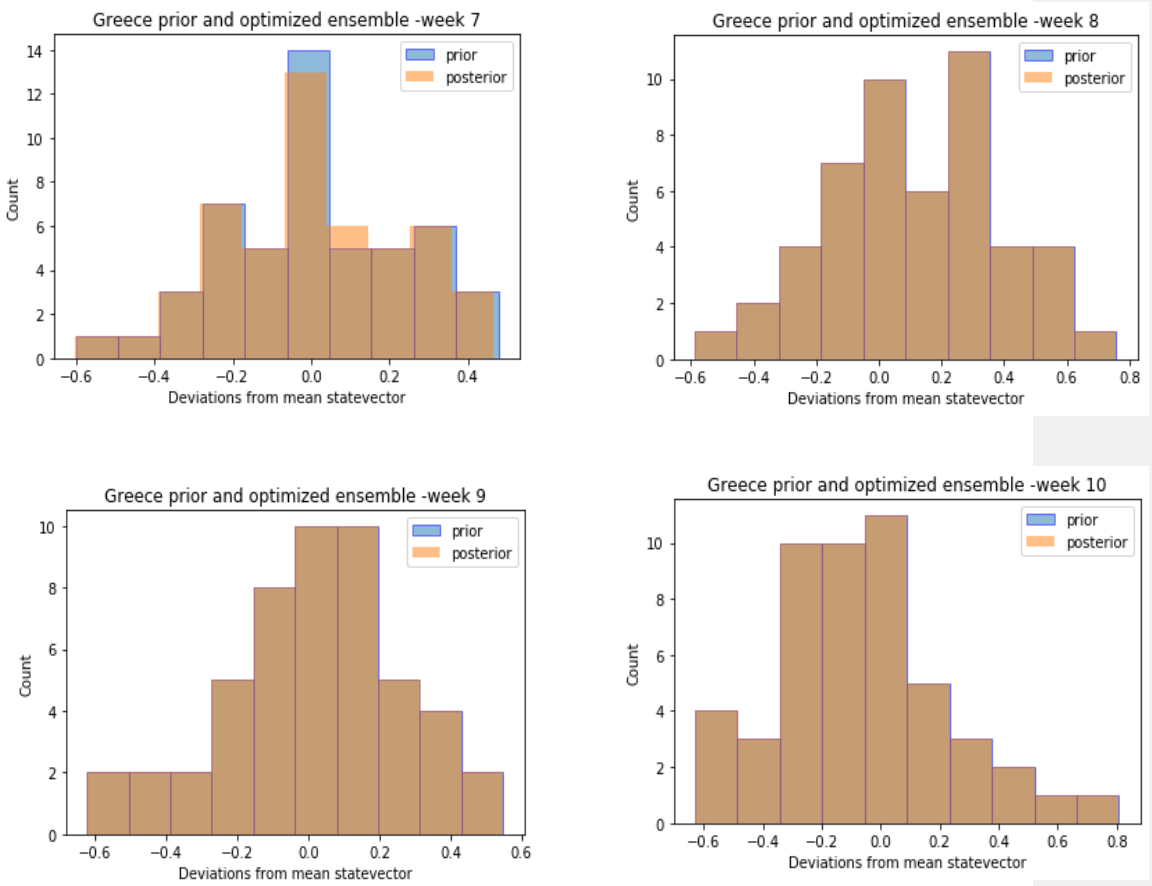


Figure 22. Prior and Posterior deviations from mean statevector λ for the region of Greece for the 10.5 run. Note the differences in the x-axis scale in the panels.

In Fig. 23, observations (blue), simulated apriori concentrations H(x) (orange) and posteriori concentrations (green) for the week 26 February - 4 March 2019 (week 9) are depicted. This week showed the most negative mean difference between observations and prior concentrations. We observe that the majority of the observations are lower than the prior concentrations, thus the posterior concentrations are lower, leading eventually to negative emission fluxes in our results. The difference between observations and prior H(x) as well as the mean of this difference for the same week are shown in Fig. 24.

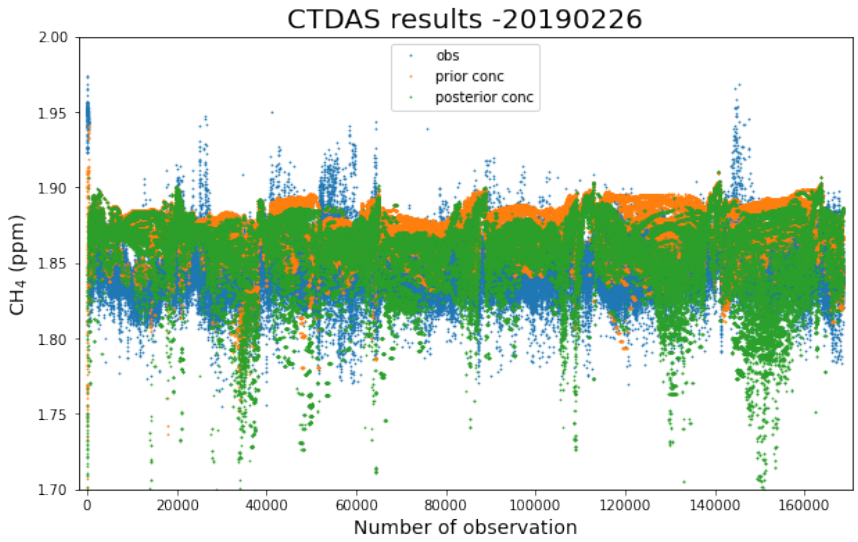


Figure 23. Methane observations from in-situ and satellite data and prior and posterior modeled concentrations for the week 26/2-4/3/2019 (week 9) (10.5 run).

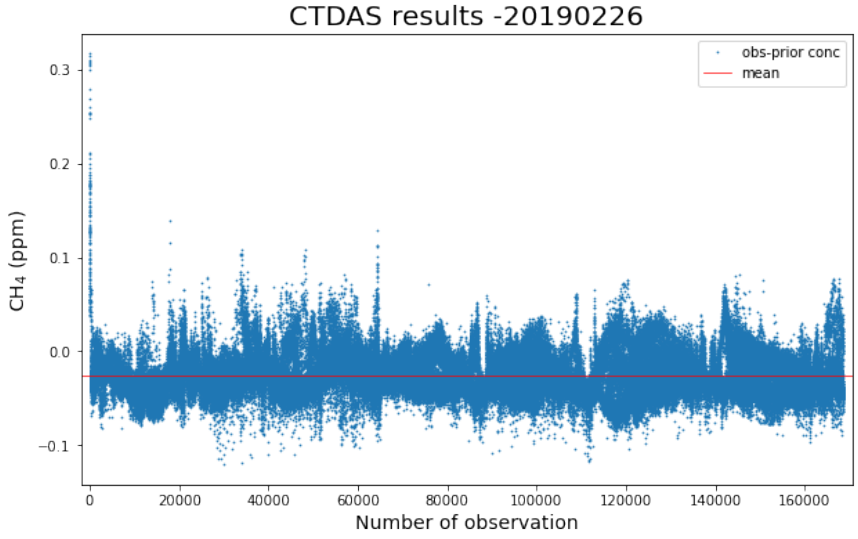
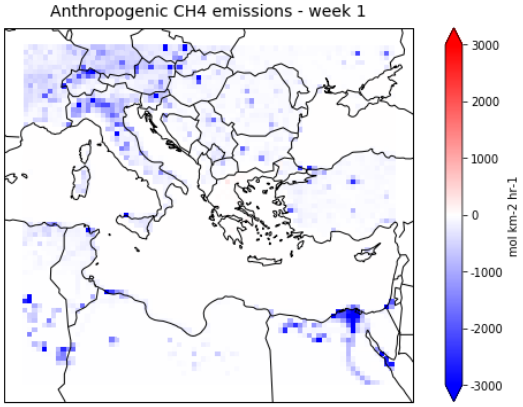


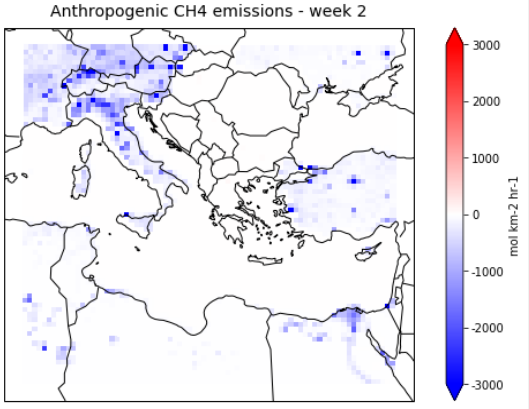
Figure 24. Difference between observed and simulated methane concentrations as well as the mean of the difference for the specific week 26/2-4/3/2019 (week 9) (10.5 run).

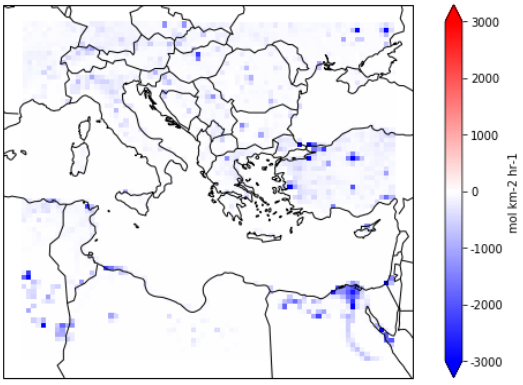
In terms of the 100 and 50 ppb run (here-after 100.50 run), the emission maps are displayed in Fig. 25. The CH_4 emissions for the finest domain are also displayed in Appendix B. Optimized emissions by the 100.50 run and the 10.5 run are different. In Table 2, the regions that have positive emissions for each simulation and each week are noted. For the ten weeks optimized, the simulation with the larger error predicts zero to two positive regions more or less than the small error simulation, except from the week 5. Since the measurement errors are the same in both inversions, this indicates that transport model error can largely impact the inversion results.

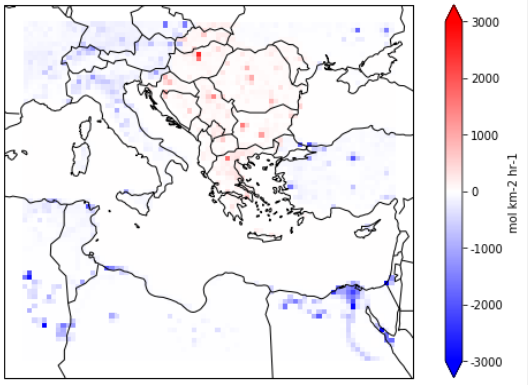
In 100.50 run, less observations are rejected compared to the 10.5 run, since model-data-mismatch R has higher values (see section 2.4.4). Improving the transport model or rejecting atmospheric observations when the corresponding footprints from the transport model are regarded untrustworthy have been the main approaches for lowering the impact of transport model error on urban inversions. Discarding atmospheric observations from inversions, on the other hand, limits the quantity of data available to constrain fluxes and can bias aggregated flux estimates (Gourdji et al., 2018).

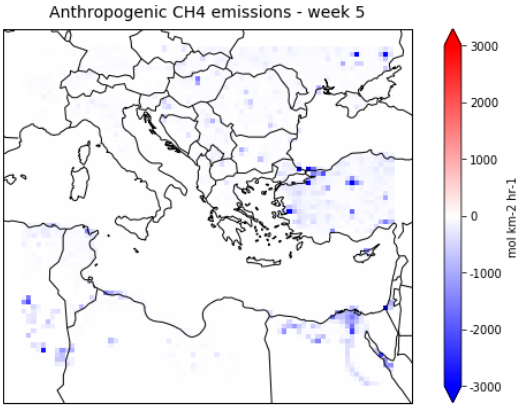
Methane observations and apriori as well as posteriori modeled concentrations for the week 9 in Fig. 26 indicate that ten times larger transport model error make a small, notable change in the simulated methane concentrations. In Fig. 27, we compare the posteriori concentrations of the two inversions and each posteriori with the observations. It can be seen that posterior concentrations for the 100.50 run are higher than the 10.5 run ones, that is closer to the apriori concentrations. This is expected since a large model-data-mismatch error will force the system to change slightly based on the observation information, thus the difference between the apriori and posteriori concentrations will be smaller.The difference between observations and modeled concentrations in Fig. 28 depict again that high boundaries result in apriori concentrations that are in average higher than the observations.

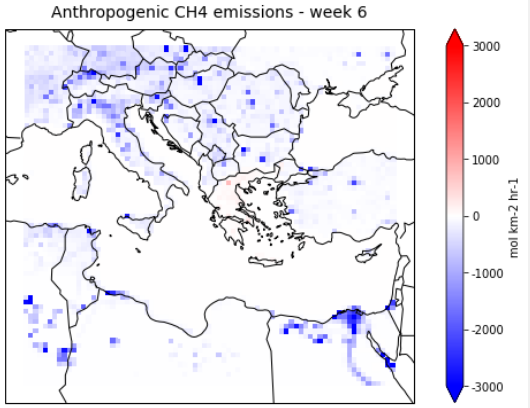

Anthropogenic CH4 emissions - week 1
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 2
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 3
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 4
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 5
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 6
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1

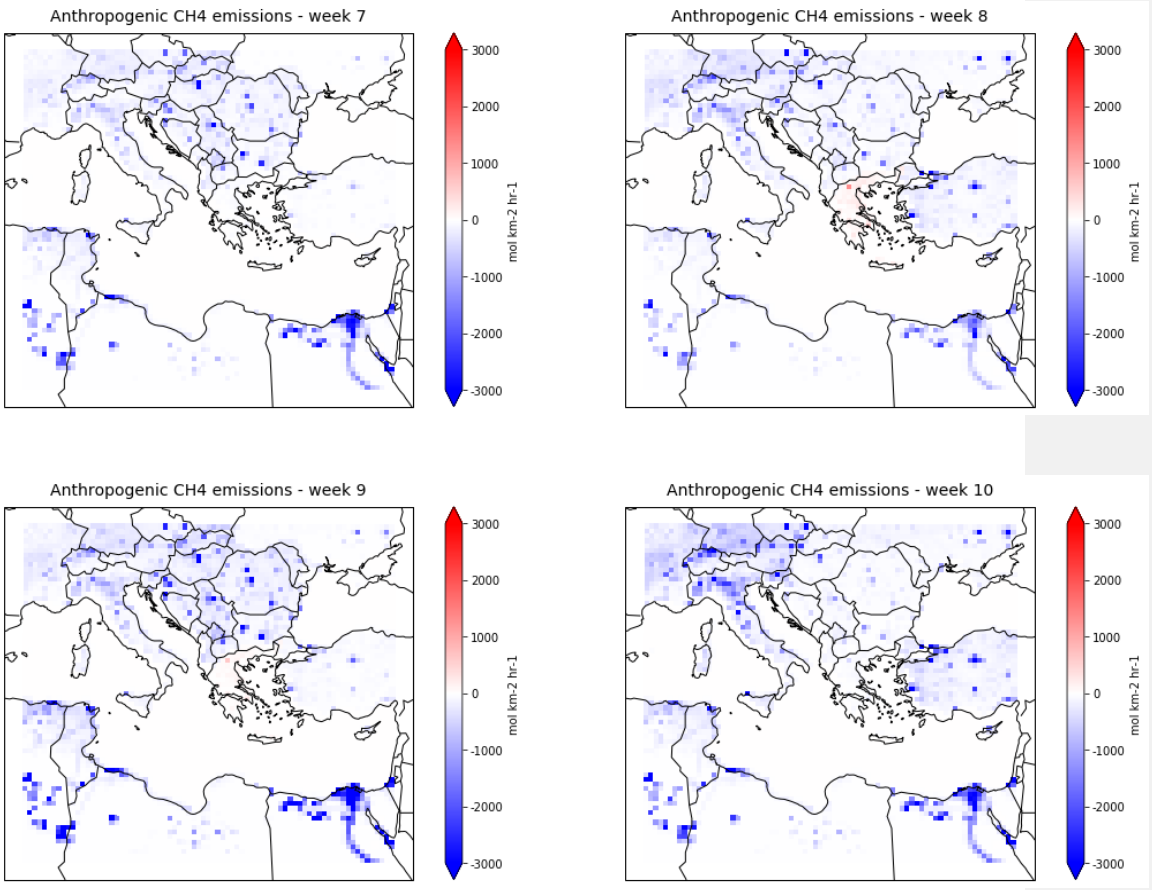


Figure 25. Optimized anthropogenic methane emissions over Eastern Mediterranean (100.50 run for the period 1/1/2019-11/3/2019).

Table 2. Regions with positive emissions for the 10.5 and for the 100.50 simulation respectively for the period 1/1/2019-11/3/2019.

Week	*10.5 run*	*100.50 run*
1	Greece, East	Greece
2	Greece	Greece, North
3	-	-
4	North	Greece, North
5	Greece, East, West, South	-
6	-	Greece
7	East, West	-
8	-	Greece
9	-	Greece
10	-	-

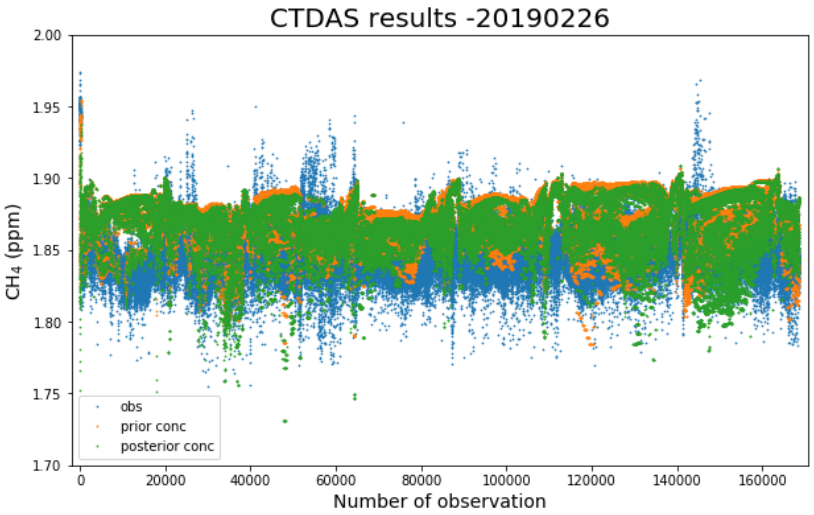


Figure 26. Methane observations from in-situ and satellite data and prior and posterior modeled concentrations for 26/2-4/3/2019 (week 9) (100.50 run)

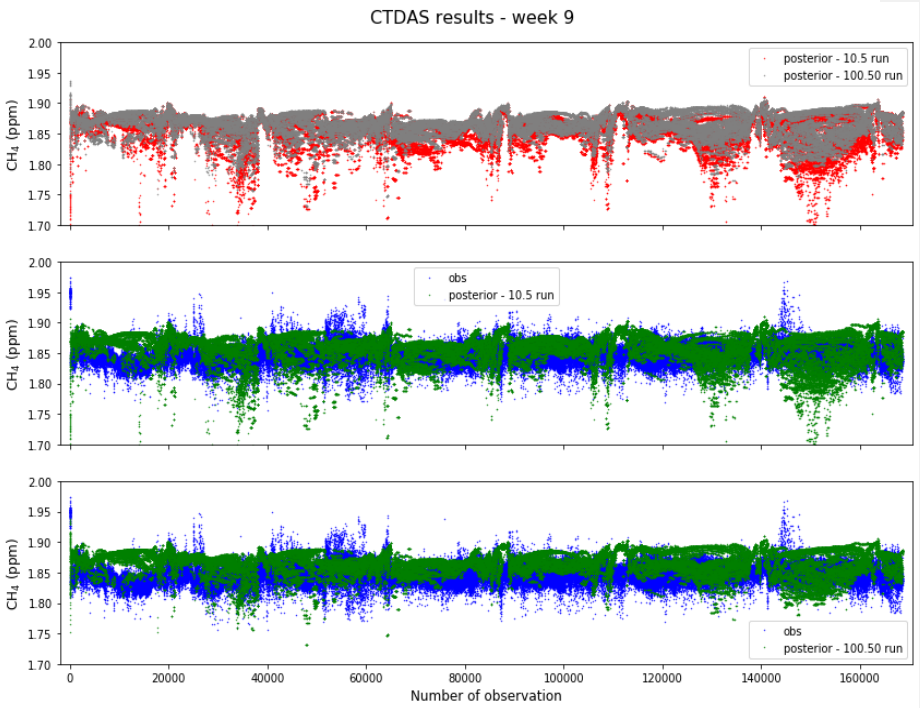


Figure 27. Posterior concentrations for the two simulations in the upper panel and comparison with the observations for week 9 of our inversion in the bottom two panels.

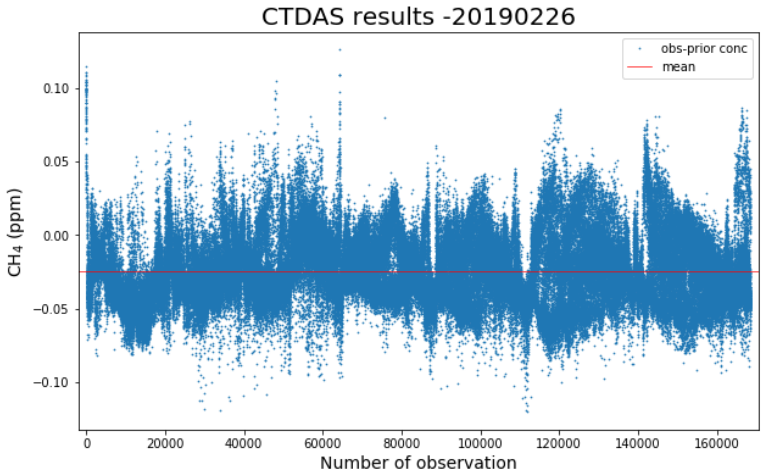


Figure 28. Difference between the observed and apriori simulated methane concentrations as well as the mean of the difference for the week 26/2-4/3/2019 (week 9) (100.50 run).

3.2 Discussion

We have conducted two inversions for optimizing methane emissions using total anthropogenic emissions of Copernicus Atmosphere Monitoring Service inventory over Central and Eastern Mediterranean, and Greece. Different transport model errors of different magnitude for TROPOMI/WFMD satellite product and in-situ measurements from Finokalia station are selected in order to examine the effect that this type of error has on our results.

In all of the weeks inverted for the two inversions, almost only negative fluxes are predicted after a period of 5 weeks. Possible reasons for that are the boundary conditions used and the lack of the major chemical sink of methane, OH, from WRF-GHG model. Therefore, methane flows out of our domains mainly depending on the meteorology. Thus, the vast majority of methane molecules tend to accumulate over the domain in general. A possible solution to this problem would be to add a methane decay factor in WRF-GHG model that agrees with methane's lifetime. However, due to the long lifetime of CH_4, of several years, this ommission is not expected to have major impact on our resutls.

In terms of boundary conditions, an alternative model, except from TM5, can be used, the simulations of which would be closer to the observations over our domain. A reanalysis product, which will be created by assimilating observations over Eastern Mediterranean, for instance, would be an appropriate constraint to the inversion. This would probably prevent the unphysical results of negative emissions. Lastly, negative emissions might also be attributed to the ill-posedness of the inversion problem. Miller et al. (2014) pointed out that inverse modelling approaches based on Gaussian assumptions, such as Ensemble Kalman filter, cannot incorporate physical bounds (e.g. non-negative emissions) and often produce unrealistic results. Therefore, it may be required to impose non-negativity constraints on the covariance matrices to ensure positive flux results.

Increasing the transport model error ten times, to 100 ppb and 50 ppb for satellite and in-situ data respectively, changes the number of regions that are predicted to have positive emission fluxes in comparison to the 10 ppb and 5 ppb run. A small model-data mismatch can either mean that the observation does not provide any constrain on the emissions or the prior emission fluxes are correct. When there is large transport error, the inversion could generate unrealistic results (negative fluxes). For observing locations in the near-field of large sources, transport model representations are likely to be much more difficult than for sites in the global network, which are often in distant areas with well-mixed air (Gourdji et al., 2018). Thus, transport model error, which is

difficult to be assumed and many times its importance is neglected in inversions, needs to be carefully estimated in order to avoid misleading inversion results.

APPENDIX A

Variability of methane concentrations over large urban agglomerations - Athens, Thessaloniki, and Patras.

For Finokalia, in-situ measurements are used. For Athens, Thessaloniki and Patras, gridded TROPOMI columns are used. Due to the long lifetime of methane, it can be inferred that the methane columnar values account for the background concentration of methane in the atmosphere. Comparing Athens, Thessaloniki, and Patras with Finokalia, we observe that the seasonality for all sites, considering the respective errors, is in general agreement. The months of March and October in Finokalia tend to be higher and lower than the other sites, respectively.The coverage of TROPOMI over the big cities is not adequate in these months, thus Finokalia data should be considered more reliable.

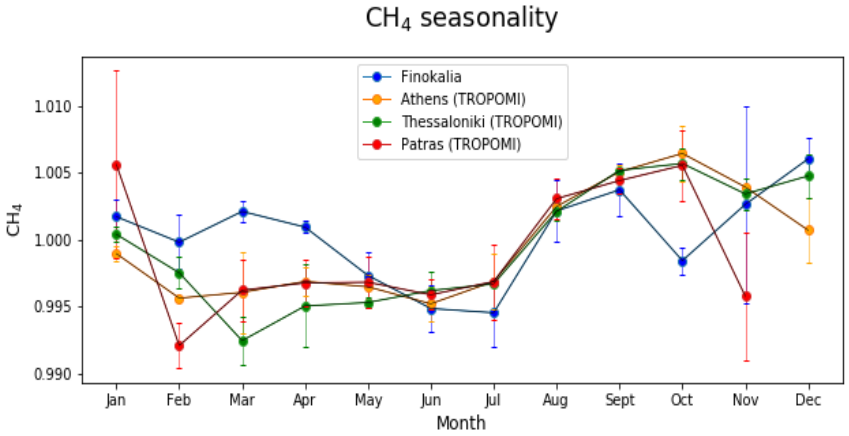
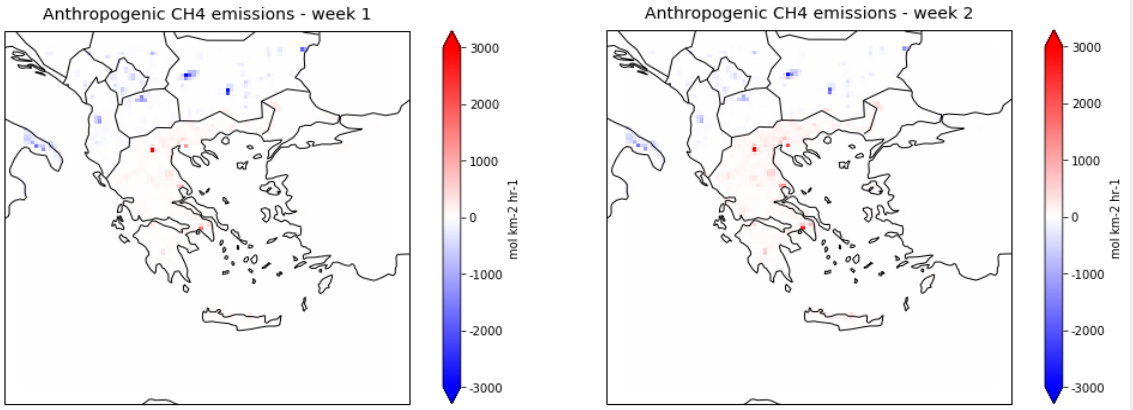
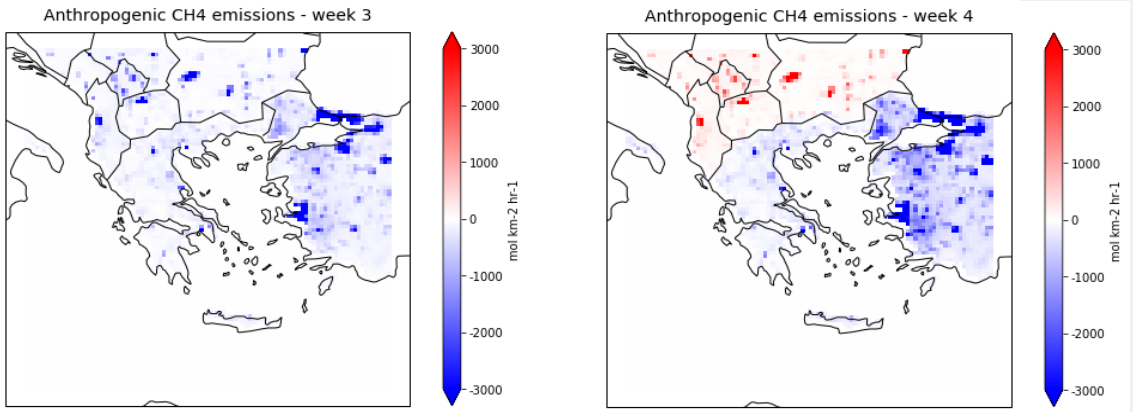


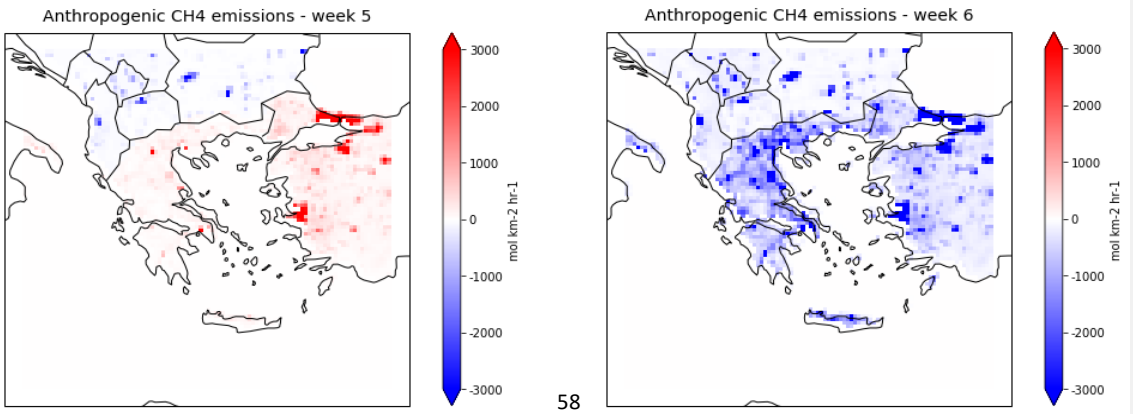
Figure A1. Methane normalized seasonality for the period 2019-2020.

APPENDIX B

Total optimized methane emissions over the finest domain over the inversion setup for the two runs with different transport model errors.







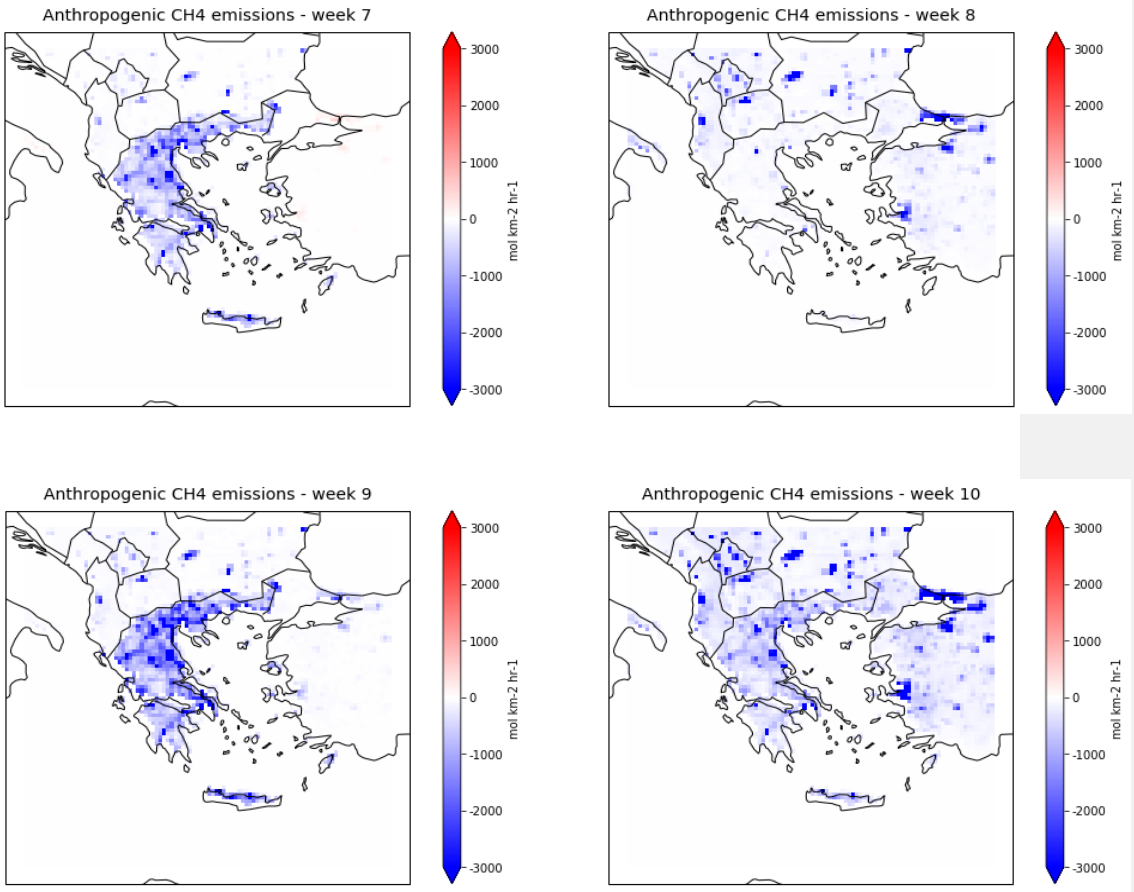
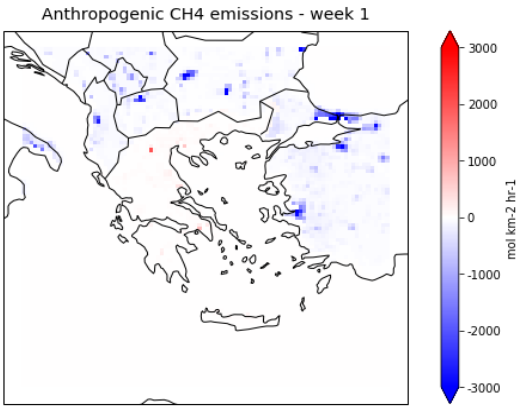
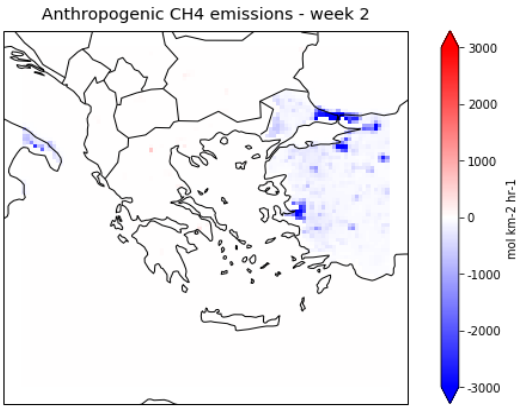
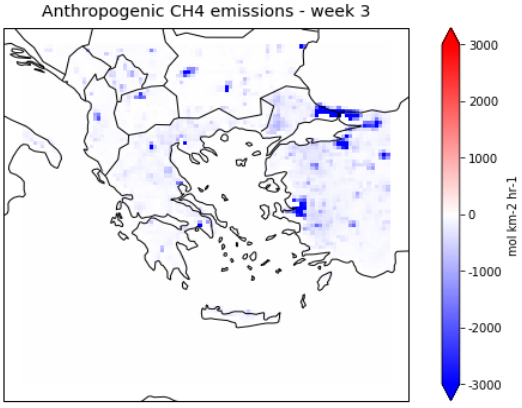
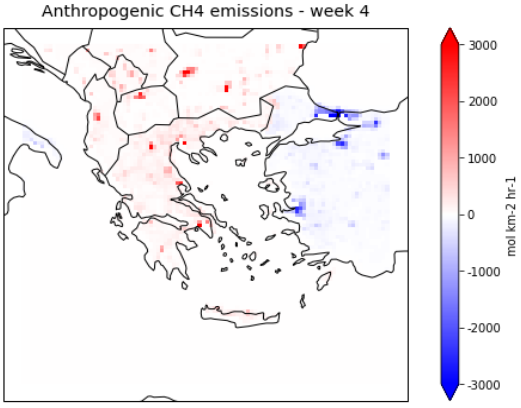


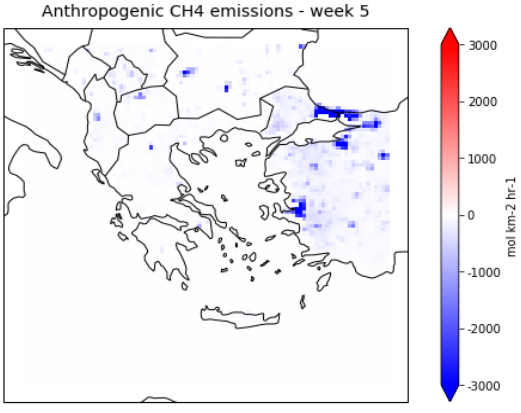
Figure B1. 10.5 run emission maps for the domain over Greece for the period 1/1/2019-11/3/2019.

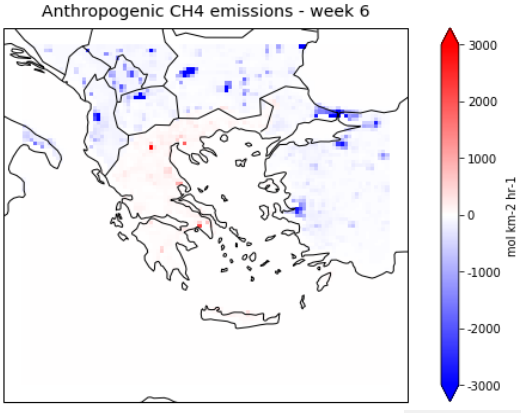

Anthropogenic CH4 emissions - week 1
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 2
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 3
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 4
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 5
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1


Anthropogenic CH4 emissions - week 6
3000
2000
1000
0
-1000
-2000
-3000
mol km-2 hr-1

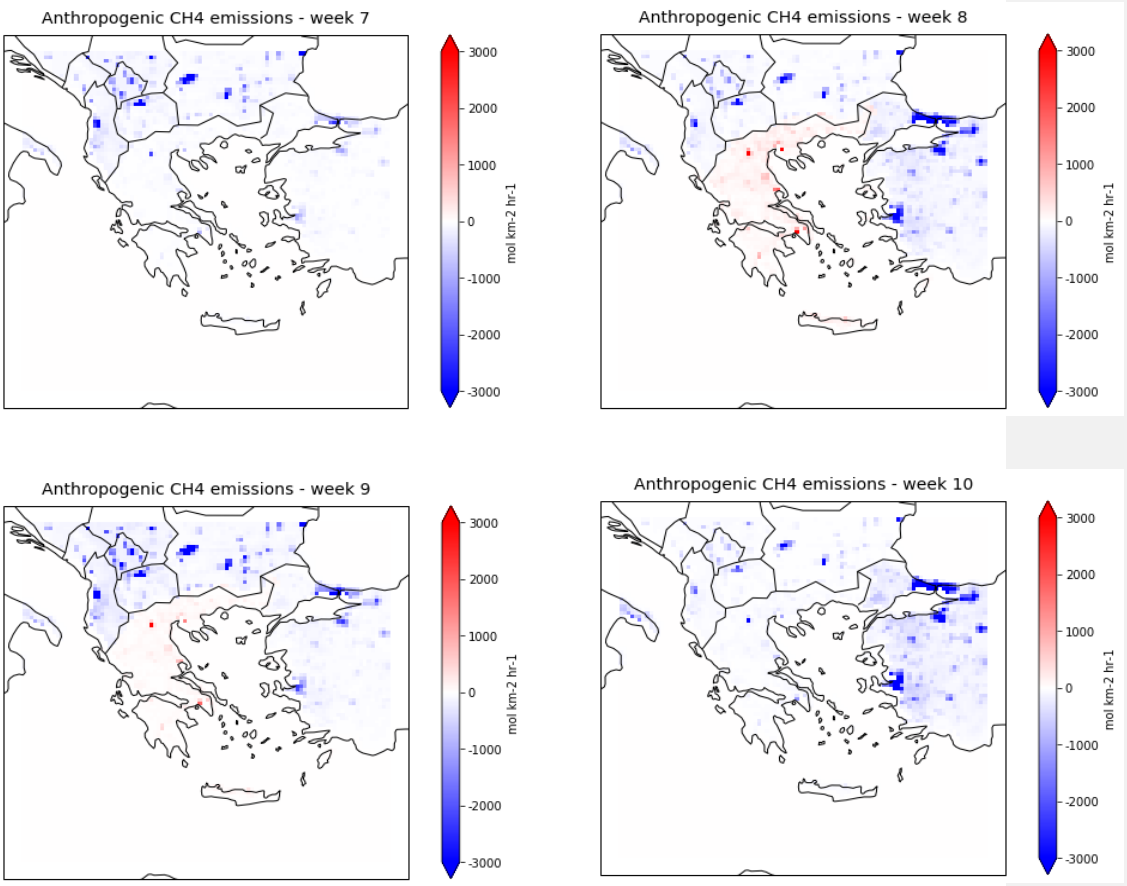


Figure B2. 100.50 run emission maps for the domain over Greece for the period 1/1/2019-11/3/2019.

APPENDIX C

Modified code for reading of TROPOMI/WFMD satellite product (obs_WRF_xch4.py) and code for the sampling of in-situ measurements (wrfout_flask_sampler.py, wrfchem_flask_helper.py).

obs_WRF_xch4.py

```
import os
import sys
import logging
import pandas as pd
import datetime as dtm
import numpy as np
from numpy import array, logical_and, sqrt
sys.path.append(os.getcwd())
sys.path.append('../../')

identifier = 'CarbonTracker total column-averaged CH4 mole fractions'
version = '0.0'

from da.observations.obs_baseclass import Observations
import da.tools.io4 as io
import da.tools.rc as rc

################### Begin Class TROPOMIObservations ###################

class TotalColumnSample(object):
    """ an object that holds data + methods and attributes needed to manipulate mole fraction values """

    def __init__(self, idx, codex, scanlen, ground, groundlen,  xdate, obs=0.0, simulated=0.0, lat=-999., lon=-999., mdm=None,  prior=0.0,  prior_profile=0.0,\
                    av_kernel=0.0,   pressure=0.0,  pressure_weighting_function=None, level_def  ="layer_average",  psurf  =  float('nan'),  resid=0.0,  hphr=0.0,  flag=0, species='ch4', sdev=0.0,\
latc_0=None,  latc_1=None,  latc_2=None,  latc_3=None,  lonc_0=None,  lonc_1=None, lonc_2=None, lonc_3=None ):

        self.id             = idx
        self.code           = codex
        self.scanlen        = scanlen
        self.ground         = ground
        self.groundlen      = groundlen
        self.xdate          = xdate
        self.obs            = obs               # Value observed
        self.simulated      = simulated         # Value simulated by model
        self.lat            = lat               # Sample lat
        self.lon            = lon               # Sample lon
        self.latc_0         = latc_0            # Sample latitude corner
        self.latc_1         = latc_1            # Sample latitude corner
        self.latc_2         = latc_2            # Sample latitude corner
        self.latc_3         = latc_3            # Sample latitude corner
        self.lonc_0         = lonc_0            # Sample longitude corner
        self.lonc_1         = lonc_1            # Sample longitude corner
        self.lonc_2         = lonc_2            # Sample longitude corner
        self.lonc_3         = lonc_3            # Sample longitude corner
        self.mdm            = mdm               # Model data mismatch
```

```
        self.prior           =  prior          #  A  priori  column  value  used  in
retrieval
        self.prior_profile  = prior_profile    # A priori profile used in retrieval
        self.av_kernel      = av_kernel         # Averaging kernel
        self.pressure       = pressure
        self.pressure_weighting_function     = pressure_weighting_function
        self.level_def      =  level_def          # Are prior and averaging kernel
defined as layer averages?
        self.psurf          = psurf             # Surface pressure (only needed if
level_def is "layer_average")
        self.loc_L          = int(0)            # localization length

        self.resid          = resid             # Mole fraction residuals
        self.hphr           = hphr                # Mole fraction prior uncertainty
from fluxes and (HPH) and model data mismatch (R)
        self.may_localize    = True              # Whether sample may be localized
in optimizer
        self.may_reject     = True              # Whether sample may be rejected if
outside threshold
        self.flag           = flag              # Flag
        self.sdev           = sdev              # standard deviation of ensemble
        self.species        = species.strip()


################### End Class TotalColumnSample ########################


#################### Begin Class TotalColumnObservations ##################

class TotalColumnObservations(Observations):

    def setup(self, dacycle):

        self.startdate = dacycle['time.sample.start']
        self.enddate   = dacycle['time.sample.end']

        sat_dirs = dacycle.dasystem['obs.column.input.dir'].split(',')
        sat_files = dacycle.dasystem['obs.column.ncfile'].split(',')

        self.sat_dirs    = []
        self.sat_files   = []
        for i in range(len(sat_dirs)):
            if not os.path.exists(sat_dirs[i].strip()):
                msg = 'Could not find the required satellite input directory (%s) '
% sat_dirs[i]
                logging.error(msg)
                raise IOError(msg)
            else:
                self.sat_dirs.append(sat_dirs[i].strip())
                self.sat_files.append(sat_files[i].strip())
        del i

        # Get observation selection criteria (if present):
        if   'obs.column.selection.variables'   in   dacycle.dasystem.keys()   and
'obs.column.selection.criteria' in dacycle.dasystem.keys():
            self.selection_vars                                                 =
dacycle.dasystem['obs.column.selection.variables'].split(',')
            self.selection_criteria                                             =
dacycle.dasystem['obs.column.selection.criteria'].split(',')
            logging.debug('Data    selection    criteria    found:    %s,    %s'
%(self.selection_vars, self.selection_criteria))
        else:
            self.selection_vars     = []
            self.selection_criteria = []
            logging.info('No data observation selection criteria found, using all
observations in file.')
```

```
100.
101.                # Model data mismatch approach
102.                self.mdm_calculation = dacycle.dasystem.get('mdm.calculation')
103.                if                      self.mdm_calculation                    in
    ['parametrization','empirical','no_transport_error']:
104.                    logging.info('Model    data    mismatch    approach    =    %s'
    %self.mdm_calculation)
105.                else:
106.                    logging.warning('No  valid  model  data  mismatch  method  found.
    Valid options are \'parametrization\', \'empirical\'. ' + \
107.                                    'Using  a  constant  estimate  for  the  model
    uncertainty of 1ppm everywhere.')
108.
109.                #  Path  to  file  with  observation  error  settings  for  column
    observations
110.                if       not       os.path.exists(dacycle.dasystem['obs.column.rc']):
    #obs.column.rc
111.                    msg = 'Could not find the required column observation .rc input
    file (%s) ' % dacycle.dasystem['obs.column.rc']
112.                    logging.error(msg)
113.                    raise IOError(msg)
114.                else:
115.                    self.obs_file = (dacycle.dasystem['obs.column.rc'])
116.
117.                self.datalist = []
118.
119.
120.                # Switch to indicate whether simulated column samples are read from
    obsOperator output,
121.                #  or  whether  the  sampling  is  done  within  CTDAS  (in  obsOperator
    class)
122.                self.sample_in_ctdas    =    dacycle.dasystem['sample.in.ctdas']    if
    'sample.in.ctdas' in dacycle.dasystem.keys() else False
123.                logging.debug('sample.in.ctdas = %s' % self.sample_in_ctdas)
124.
125.
126.            def get_samples_type(self):
127.                return 'column'
128.
129.
130.            def add_observations(self):
131.                """  Reading  of  total  column  observations,  and  selection  of
    observations that will be sampled and assimilated.
132.
133.                """
134.
135.                # Read observations from daily input files
136.                for i in range(len(self.sat_dirs)):
137.
138.                    logging.info('Reading       observations       from       %s%s'
    %(self.sat_dirs[i],self.sat_files[i]))
139.
140.                    infile0 = os.path.join(self.sat_dirs[i], self.sat_files[i])
141.                    ndays = 0
142.
143.                    while self.startdate+dtm.timedelta(days=ndays) <= self.enddate:
144.
145.                        infile                                                  =
    infile0.replace("<YYYYMMDD>",(self.startdate+dtm.timedelta(days=ndays)).strftime("%
    Y%m%d"))
146.                       # d1 = infile.split('_')[8]
147.                        #dtseries = d1[0:-3]
148.
149.                        if os.path.exists(infile):
150.                            logging.info("Reading    observations    for    %s"    %
    (self.startdate+dtm.timedelta(days=ndays)).strftime("%Y%m%d"))
```

```
151.                             len_init = len(self.datalist)
152.
153.
154.                             # get index of observations that satisfy selection
    criteria (based on variable names and values in system rc file, if present)
155.                             ncf    = io.ct_read(infile, 'read')
156.
157.                             if self.selection_vars:
158.                                 selvars = []
159.                                 for j in self.selection_vars:
160.                                     selvars.append(ncf.get_variable(j.strip()))
161.                                 del j
162.                                 criteria = []
163.                                 for j in range(len(self.selection_vars)):
164.                                     criteria.append(eval('selvars[j]'+self.selection
    _criteria[j]))
165.                                 del j
166.                                 subselect                                          =
    np.logical_and.reduce(criteria).nonzero()[0]
167.                             else:
168.                                 subselect                                          =
    np.arange(ncf.get_variable('sounding_id').size)
169.
170.
171.                             code      = ncf.get_attribute('tracking_id')
172.                             level_def = "layer_average"
173.                              # read observations
174.                             ids                                                    =
    ncf.get_variable('sounding_id').take(subselect,axis=0)
175.                             scanlen       = len(ids)
176.                             ground                                                 =
    ncf.get_variable('ground_pixel').take(subselect,axis=0)
177.                             groundlen     = len(ground)
178.                             lats                                                   =
    ncf.get_variable('latitude').take(subselect,axis=0)
179.                             lons                                                   =
    ncf.get_variable('longitude').take(subselect,axis=0)
180.                             obs                                                    =
    ncf.get_variable('xch4').take(subselect,axis=0)
181.                             unc                                                    =
    ncf.get_variable('xch4_uncertainty').take(subselect,axis=0)
182.                             dates                                                  =
    ncf.get_variable('time').take(subselect,axis=0)
183.                             dates         = array([dtm.datetime.fromtimestamp(d) for
    d in dates]
184.                             av_kernel                                              =
    ncf.get_variable('xch4_averaging_kernel').take(subselect,axis=0)
185.                             prior_profile                                          =
    ncf.get_variable('ch4_profile_apriori').take(subselect,axis=0)
186.                             pressure                                               =
    ncf.get_variable('pressure_levels').take(subselect,axis=0)
187.                             prior         = [float('nan')]*len(ids)
188.                             pwf                                                    =
    ncf.get_variable('pressure_weight').take(subselect,axis=0)
189.                             psurf         =  [float('nan')]*len(ids)
190.
191.                             # Optional: footprint corners
192.                             latc = dict(
193.                                 latc_0=[float('nan')]*len(ids),
194.                                 latc_1=[float('nan')]*len(ids),
195.                                 latc_2=[float('nan')]*len(ids),
196.                                 latc_3=[float('nan')]*len(ids))
197.                             lonc = dict(
198.                                 lonc_0=[float('nan')]*len(ids),
199.                                 lonc_1=[float('nan')]*len(ids),
200.                                 lonc_2=[float('nan')]*len(ids),
```

```
                        lonc_3=[float('nan')]*len(ids))

                    ncf.close()

                    # Add samples to datalist
                    # Note that the mdm is initialized here equal to the measurement uncertainty. This value is used in add_model_data_mismatch to calculate the mdm including model error
                    logging.info("Size scan groun dates obs lats lons av_kern %d %d %d %d %d %d %d " % (ids.size,ground.size,dates.size,obs.size,lats.size,lons.size,av_kernel.size))
                    for k in range(len(ids)):
                        # Check for every sounding if time is between start and end time (relevant for first and last days of window)
                        if self.startdate <= dates[k] <= self.enddate:
                            self.datalist.append(TotalColumnSample(ids[k], code, scanlen, ground[k], groundlen, dates[k], obs[k]* 1.e-3, None, lats[k], lons[k], unc[k]* 1.e-3,prior=prior[k], prior_profile=prior_profile[k,:]* 1.e-3,\
                            av_kernel=av_kernel[k,:], pressure=pressure[k,:], pressure_weighting_function=pwf[k,:], level_def=level_def, psurf=psurf[k], latc_0=latc['latc_0'][k], latc_1=latc['latc_1'][k], latc_2=latc['latc_2'][k], latc_3=latc['latc_3'][k],
                            lonc_0=lonc['lonc_0'][k], lonc_1=lonc['lonc_1'][k], lonc_2=lonc['lonc_2'][k], lonc_3=lonc['lonc_3'][k]
  ))

                    logging.debug("Added %d observations to the Data list" % (len(self.datalist)-len_init))

                ndays += 1

        del i

        if len(self.datalist) > 0:
            logging.info("Observations list now holds %d values" % len(self.datalist))
        else:
            logging.info("No observations found for sampling window")


    def add_model_data_mismatch(self, filename=None, advance=False):

        obs_data = rc.read(self.obs_file)
        self.rejection_threshold = int(obs_data['obs.rejection.threshold'])

        # At this point mdm is set to the measurement uncertainty only, added in the add_observations function.
        # Here this value is used to set the combined mdm by adding an estimate for the model uncertainty as a sum of squares.
        if len(self.datalist) <= 1: return
        for obs in self.datalist:
            # parametrization as used by Frederic Chevallier
            if self.mdm_calculation == 'parametrization':
                obs.mdm = ( obs.mdm*obs.mdm + (0.8*np.exp((90.0+obs.lat)/300.0))**2 )**0.5
            # empirical approach of Andy Jacobson, TO BE IMPLEMENTED
            # elif self.mdm_calculation == 'empirical':
            #     obs.mdm = ...
            elif self.mdm_calculation == 'no_transport_error':
                pass
            else: # assume general model uncertainty of 1 ppm (arbitrary value)
                obs.mdm = ( obs.mdm*obs.mdm + 0.2**2 )**0.5
        del obs

```

```
249.                meanmdm = np.average(np.array( [obs.mdm for obs in self.datalist] ))
250.                logging.debug('Mean MDM = %s' %meanmdm)
251.
252.
253.            def add_simulations(self, filename, silent=False):
254.                """ Adds observed and model simulated column values to the mole
    fraction objects
255.                    This    function    includes    the    add_observations    and
    add_model_data_mismatch functionality for the sake of computational efficiency
256.
257.                """
258.
259.                if self.sample_in_ctdas:
260.                    logging.debug("CODE TO ADD SIMULATED SAMPLES TO DATALIST TO BE
    ADDED")
261.
262.                else:
263.                    # read simulated samples from file
264.                    if not os.path.exists(filename):
265.                        msg = "Sample output filename for observations could not be
    found : %s" % filename
266.                        logging.error(msg)
267.                        logging.error("Did the sampling step succeed?")
268.                        logging.error("...exiting")
269.                        raise IOError(msg)
270.
271.                    ncf       = io.ct_read(filename, method='read')
272.                    logging.debug("Read  Observed  and  Simulated  SAMPLES  from  file
    (%s)" % filename)
273.                    ids       = ncf.get_variable('sounding_id')
274.                    simulated = ncf.get_variable('column_modeled')
275.                    ncf.close()
276.                    logging.info("Successfully  read  data  from  model  sample  file
    (%s)" % filename)
277.
278.                    obs_ids = self.getvalues('id').tolist()
279.
280.                    missing_samples = []
281.
282.                    # Match read simulated samples with observations in datalist
283.                    logging.info("Adding %i simulated samples to the data list..." %
    len(ids))
284.                    for i in range(len(ids)):
285.                        # Assume samples are in same order in both datalist and file
    with simulated samples...
286.                        if ids[i] == obs_ids[i]:
287.                            self.datalist[i].simulated = simulated[i]
288.                        # If not, find index of current sample
289.                        elif ids[i] in obs_ids:
290.                            index = obs_ids.index(ids[i])
291.                            # Only add simulated value to datalist if sample has not
    been filled before. Otherwise: exiting
292.                            if self.datalist[index].simulated is not None:
293.                                msg = 'Simulated and observed samples not in same
    order, and duplicate sample IDs found.'
294.                                logging.error(msg)
295.                                raise IOError(msg)
296.                            else:
297.                                self.datalist[index].simulated = simulated[i]
298.                        else:
299.                            logging.debug('added %s to missing_samples, obs id = %s'
    %(ids[i],obs_ids[i]))
300.                            missing_samples.append(ids[i])
301.                    del i
302.
303.                    if not silent and missing_samples != []:
```

```
304.                     logging.warning('%i Model samples were found that did not
    match any ID in the observation list. Skipping them...' % len(missing_samples))
305.
306.                 # if number of simulated samples < observations: remove
    observations without samples
307.                 if len(simulated) < len(self.datalist):
308.                     test = len(self.datalist) - len(simulated)
309.                     logging.warning('%i Observations were not sampled, removing
    them from datalist...' % test)
310.                     for index in reversed(list(range(len(self.datalist)))):
311.                         if self.datalist[index].simulated is None:
312.                             del self.datalist[index]
313.                     del index
314.
315.                 logging.debug("%d simulated values were added to the data list"
    % (len(ids) - len(missing_samples)))
316.
317.         def write_sample_coords(self, obsinputfile):
318.             """
319.                 Write empty sample_coords_file if soundings are present in time
    interval, just such that general pipeline code does not have to be changed...
320.             """
321.
322.             if self.sample_in_ctdas:
323.                 return
324.
325.             if len(self.datalist) <= 1: #== 0:
326.                 logging.info("No observations found for this time period, no obs
    file written")
327.                 return
328.
329.             # write data required by observation operator for sampling to file
330.             f = io.CT_CDF(obsinputfile, method='create')
331.             logging.debug('Creating      new      observations      file      for
    ObservationOperator       (%s)       containing       %d       observations'       %
    (obsinputfile,len(self.datalist)))
332.
333.             dimsoundings  = f.add_dim('soundings',len(self.datalist))
334.             dimepoch      = f.add_dim('epoch_dimension', 7)
335.             dimchar      = f.add_dim('char', 30)
336.
337.             if len(self.datalist) == 1:
338.                 dimlevels = f.add_dim('levels', len(self.getvalues('pressure')))
339.             else:
340.                 dimlevels                  =                  f.add_dim('levels',
    self.getvalues('pressure').shape[1])
341.
342.             if len(self.datalist) == 1:
343.                 dimlayers = f.add_dim('layers',len(self.getvalues('av_kernel')))
344.             else:
345.                 dimlayers                  =                  f.add_dim('layers',
    self.getvalues('av_kernel').shape[1])
346.
347.
348.             savedict           = io.std_savedict.copy()
349.             savedict['dtype']  = "int64"
350.             savedict['name']   = "sounding_id"
351.             savedict['dims']   = dimsoundings
352.             savedict['values'] = self.getvalues('id').tolist()
353.             f.add_data(savedict)
354.
355.
356.             data  =  [[d.year,  d.month,  d.day,  d.hour,  d.minute,  d.second,
    d.microsecond] for d in self.getvalues('xdate') ]
357.             savedict           = io.std_savedict.copy()
358.             savedict['dtype']  = "int"
```

```
359.                savedict['name']   = "date"
360.                savedict['dims']   = dimsoundings + dimepoch
361.                savedict['values'] = data
362.                f.add_data(savedict)
363.
364.                savedict           = io.std_savedict.copy()
365.                savedict['name']   = "latitude"
366.                savedict['dims']   = dimsoundings
367.                savedict['values'] = self.getvalues('lat').tolist()
368.                f.add_data(savedict)
369.
370.                savedict           = io.std_savedict.copy()
371.                savedict['name']   = "longitude"
372.                savedict['dims']   = dimsoundings
373.                savedict['values'] = self.getvalues('lon').tolist()
374.                f.add_data(savedict)
375.
376.                savedict = io.std_savedict.copy()
377.                savedict['name']   = "averaging_kernel"
378.                savedict['dims']   = dimsoundings + dimlayers
379.                savedict['values'] = self.getvalues('av_kernel').tolist()
380.                f.add_data(savedict)
381.
382.                savedict = io.std_savedict.copy()
383.                savedict['name']   = "prior_profile"
384.                savedict['dims']   = dimsoundings + dimlayers
385.                savedict['missing_value'] = "-999999."
386.                savedict['values'] = self.getvalues('prior_profile').tolist()
387.                f.add_data(savedict)
388.
389.
390.                savedict = io.std_savedict.copy()
391.                savedict['name']   = "prior"
392.                savedict['dims']   = dimsoundings
393.                savedict['values'] = self.getvalues('prior').tolist()
394.                f.add_data(savedict)
395.
396.
397.                savedict = io.std_savedict.copy()
398.                savedict['name']   = "psurf"
399.                savedict['dims']   = dimsoundings
400.                savedict['values'] = self.getvalues('psurf').tolist()
401.                f.add_data(savedict)
402.
403.                savedict = io.std_savedict.copy()
404.                savedict['name']   = "pressure_levels"
405.                savedict['dims']   = dimsoundings + dimlevels
406.                savedict['values'] = self.getvalues('pressure').tolist()
407.                f.add_data(savedict)
408.
409.                savedict = io.std_savedict.copy()
410.                savedict['name']   = "pressure_weighting_function"
411.                savedict['dims']   = dimsoundings + dimlayers
412.                savedict['values']                                          =
    self.getvalues('pressure_weighting_function').tolist()
413.                f.add_data(savedict)
414.
415.                savedict           = io.std_savedict.copy()
416.                savedict['name']   = "latc_0"
417.                savedict['dims']   = dimsoundings
418.                savedict['values'] = self.getvalues('latc_0').tolist()
419.                f.add_data(savedict)
420.
421.                savedict           = io.std_savedict.copy()
422.                savedict['name']   = "latc_1"
423.                savedict['dims']   = dimsoundings
```

```
424.                savedict['values'] = self.getvalues('latc_1').tolist()
425.                f.add_data(savedict)
426.
427.                savedict           = io.std_savedict.copy()
428.                savedict['name']   = "latc_2"
429.                savedict['dims']   = dimsoundings
430.                savedict['values'] = self.getvalues('latc_2').tolist()
431.                f.add_data(savedict)
432.
433.                savedict           = io.std_savedict.copy()
434.                savedict['name']   = "latc_3"
435.                savedict['dims']   = dimsoundings
436.                savedict['values'] = self.getvalues('latc_3').tolist()
437.                f.add_data(savedict)
438.
439.                savedict           = io.std_savedict.copy()
440.                savedict['name']   = "lonc_0"
441.                savedict['dims']   = dimsoundings
442.                savedict['values'] = self.getvalues('lonc_0').tolist()
443.                f.add_data(savedict)
444.
445.                savedict           = io.std_savedict.copy()
446.                savedict['name']   = "lonc_1"
447.                savedict['dims']   = dimsoundings
448.                savedict['values'] = self.getvalues('lonc_1').tolist()
449.                f.add_data(savedict)
450.
451.                savedict           = io.std_savedict.copy()
452.                savedict['name']   = "lonc_2"
453.                savedict['dims']   = dimsoundings
454.                savedict['values'] = self.getvalues('lonc_2').tolist()
455.                f.add_data(savedict)
456.
457.                savedict           = io.std_savedict.copy()
458.                savedict['name']   = "lonc_3"
459.                savedict['dims']   = dimsoundings
460.                savedict['values'] = self.getvalues('lonc_3').tolist()
461.                f.add_data(savedict)
462.
463.
464.                savedict = io.std_savedict.copy()
465.                savedict['dtype'] = "char"
466.                savedict['name']  = "level_def"
467.                savedict['dims']  = dimsoundings + dimchar
468.                savedict['values'] = self.getvalues('level_def').tolist()
469.                f.add_data(savedict)
470.
471.                f.close()
472.        ################### End Class TotalColumnObservations ###################
473.
474.        if __name__ == "__main__":
475.            pass
```

wrfout_flask_sampler.py

```
import os
import sys
import copy
import numpy as np
import netCDF4 as nc

# Import some CTDAS tools
pd = os.path.pardir
inc_path = os.path.join(os.path.dirname(os.path.abspath(__file__)),
                        pd, pd, pd)
inc_path = os.path.abspath(inc_path)
sys.path.append(inc_path)
from da.tools.wrfchem.wrfchem_flask_helper import WRFChemHelper_flask
from da.tools.wrfchem.utilities import utilities
import argparse

########## Parse options
parser = argparse.ArgumentParser()
parser.add_argument("--nproc", type=int,
                    help="ID of this sampling process (0 ... nprocs-1)")
parser.add_argument("--nprocs", type=int,
                    help="Number of sampling processes")
parser.add_argument("--sampling_coords_file", type=str,
                    help="File with sampling coordinates as created " + \
                         "by CTDAS column samples object")
parser.add_argument("--run_dir", type=str,
                    help="Directory with wrfout files")
parser.add_argument("--original_save_suffix", type=str,
                    help="Just leave this on .original")
parser.add_argument("--nmembers", type=int,
                    help="Number of tracer ensemble members")
parser.add_argument("--tracer_optim", type=str,
                    help="Tracer that was optimized (e.g. CO2 for " + \
                         "ensemble members CO2_000 etc.)")
parser.add_argument("--outfile_prefix", type=str,
                    help="One process: output file. More processes: " + \
                         "output file is <outfile_prefix>.<nproc>.slice")
parser.add_argument("--footprint_samples_dim", type=int,
                    help="Sample column footprint at n x n points")

args = parser.parse_args()
settings = copy.deepcopy(vars(args))

wd = os.getcwd()
try:
    os.makedirs("log")
except OSError:
    pass

logfile = os.path.join(wd, "log/wrfout_sampler." + str(settings['nproc']) + ".log")

os.system("touch " + logfile)
os.system("rm " + logfile)
os.system("echo 'Process " + str(settings['nproc']) + " of " +
str(settings['nprocs']) + ": start' >> " + logfile)
os.system("date >> " + logfile)

########## Initialize wrfhelper
wrfhelper = WRFChemHelper_flask(settings)
wrfhelper.validate_settings(['sampling_coords_file',
                             'run_dir',
                             'nproc',
                             'nprocs',
```

```
                                'original_save_suffix',  #  necessary  for  selecting
    filename
                                'nmembers', # special case 0: sample 'tracer_optim'
                                'tracer_optim',
                                'outfile_prefix',
                                'footprint_samples_dim'])

cwd = os.getcwd()
os.chdir(wrfhelper.settings['run_dir'])

#wrfhelper.namelist = wrfhelper.read_namelist(wrfhelper.settings['run_dir'])
wrfhelper.namelist = wrfhelper.read_namelist(".")


########## Figure out which samples to process
# Get number of samples
ncf = nc.Dataset(settings['sampling_coords_file'], "r")
nsamples = len(ncf.dimensions["obs"])
ncf.close()

id0,      id1      =      utilities.get_slicing_ids(nsamples,      settings['nproc'],
    settings['nprocs'])

os.system("echo 'id0=" + str(id0) + "' >> " + logfile)
os.system("echo 'id1=" + str(id1) + "' >> " + logfile)

########## Read samples from coord file
dat = wrfhelper.read_sampling_coords(settings['sampling_coords_file'], id0, id1)

os.system("echo 'Data read, len=" + str(len(dat['obs'])) + "' >> " + logfile)


########## Locate samples in wrf domains

# Take care of special case without ensemble
nmembers = settings['nmembers']
if nmembers == 0:
    # Special case: sample 'tracer_optim', don't add member suffix
    member_names = [settings['tracer_optim']]
          nmembers = 1
      else:
          member_names  =  [settings['tracer_optim']  +  "_%03d"  %  nm  for  nm  in
    range(nmembers)]


      # Keep a description of a small wrf file for each domain in memory to
      # locate observations.
      # This concept is probably obsolete - doesn't save time, and
      # locate_domain is parallelized anyway
      wrfhelper.open_wrf_location_files()

      if settings["footprint_samples_dim"]==1:
          # Locate all observations in space
          # This function Wouldn't work for moving nests.
          id_xy_f,    domain,    z    =    wrfhelper.locate_domain(dat['latitude'],
    dat['longitude'], dat['altitude'])
          # Assume box averages and don't interpolate horizontally
          id_xy = np.round(id_xy_f).astype(int)
          os.system("echo 'Domains located from obs " + str(domain) + "' >> " +
    logfile)
      else:
          # Return the whole thing (needed in wrfhelper.sample_total_columns)
          raise                    NotImplementedError("To                    do:
    wrfhelper.get_footprint_sampling_points")
          dat_fs = wrfhelper.get_footprint_sampling_points(dat)
          # Locate (free)
```

```
            id_xy_f_free, domain_free = wrfhelper.locate_domain(dat_fs['latitude'], dat_fs['longitude'])
            id_xy_free = np.round(id_xy_f_free).astype(int)
            # Determine max domain
            domain_fs = None
            raise NotImplementedError("To do: domain_fs")
            # Sample again with domain restriction - no need to return it again
            id_xy_f,       _       =       wrfhelper.locate_domain(dat_fs['latitude'], dat_fs['longitude'], domain_fs)
            id_xy = np.round(id_xy_f).astype(int)
            # Thin out domain_fs to pass it to determination of id_t and frac_t below
            domain = domain_fs[::settings["footprint_samples_dim"]]

        wrfhelper.close_wrf_location_files()

        id_t = np.zeros_like(domain)
        frac_t = np.ndarray(id_t.shape, float)
        frac_t[:] = float("nan")

        wrfout_files = dict()
        wrfout_times = dict()
        wrfout_start_time_ids = dict()

        UD = list(set(domain))
        os.system("echo 'domains " + str(UD) + "' >> " + logfile)

        for dom in UD:
            os.system("echo 'Processing domain " + str(dom) + "' >> " + logfile)
            idd = np.where(domain == dom)[0]
            os.system("echo 'idd " + str(idd) + "' >> " + logfile)
            # Get full time vector
            wrfout_files[dom]   =   wrfhelper.get_wrf_filenames("wrfout_d%02d_*00"   % dom)
            os.system("echo 'Wrf filenames  " + str(wrfout_files[dom])  + "' >> " + logfile)
            wrfout_times[dom],              wrfout_start_time_ids[dom]              = wrfhelper.wrf_times(wrfout_files[dom])

            # time id
            for idd_ in idd:
                # Look where it sorts in
                tmp = [i
                       for i in range(len(wrfout_times[dom])-1)
                       if wrfout_times[dom][i] <= dat['time'][idd_] \
                          and dat['time'][idd_] < wrfout_times[dom][i+1]]
                # Catch the case that the observation took place exactly at the
                # last timestep
                if len(tmp) == 1:
                    id_t[idd_] = tmp[0]
                    time0 = wrfout_times[dom][id_t[idd_]]
                    time1 = wrfout_times[dom][id_t[idd_]+1]
                    frac_t[idd_]  =  (time1  -  dat['time'][idd_]).total_seconds()  / (time1 - time0).total_seconds()
                    os.system("echo  'frac_t   "  +  str(frac_t[idd_])   +  "'  >>  "  +  logfile)
                else: # len must be 0 in this case
                    if len(tmp) > 1:
                        os.system("echo 'wat' >> " + logfile)
                        raise ValueError("wat")
                    if dat['time'][idd_] == wrfout_times[dom][-1]:
                        id_t[idd_] = len(wrfout_times[dom])-1
                        frac_t[idd_] = 1
                    else:
                        msg  =  "Sample  %d,  obs_num  %s:  outside  of  simulated time."%(idd_, dat['obs_num'][idd_])
```

```
                        os.system("echo '" + msg + "' >> " + logfile)
                        raise ValueError(msg)


        # Now read the data
        # Input: id_xy, dom, id_t, wrfout_start_time_ids, fract_t
        # Output: sampled columns
        # All input related to location:
        if settings["footprint_samples_dim"]>1:
            domain = domain_fs
            id_t = np.repeat(id_t, settings["footprint_samples_dim"])
            frac_t = np.repeat(frac_t, settings["footprint_samples_dim"])

        loc_input = dict(id_xy=id_xy, domain=domain,
                         id_t=id_t, frac_t=frac_t,
                         files=wrfout_files,
    file_start_time_indices=wrfout_start_time_ids, z=z)


        ens_sim = wrfhelper.sample_flask(dat, loc_input, member_names)

        # Write results to file
        obs_ids = dat['obs_num']
        # Remove simulations that are nan (=not in domain)
        if ens_sim.shape[0] > 0:
            valid = np.apply_along_axis(lambda arr: not np.any(np.isnan(arr)), 1,
    ens_sim)
            obs_ids_write = obs_ids[valid]
            ens_sim_write = ens_sim[valid, :]
        else:
            obs_ids_write = obs_ids
            ens_sim_write = ens_sim

        if settings['nprocs'] == 1:
            outfile = settings['outfile_prefix']
        else:
            # Create output files with the appendix ".<nproc>.slice"
            # Format <nproc> so that they can later be easily sorted.
            len_nproc = int(np.floor(np.log10(settings['nprocs']))) + 1
            outfile = settings['outfile_prefix'] + (".%0" + str(len_nproc) +
    "d.slice") % settings['nproc']

        os.system("echo 'Writing output file '" +
    os.path.join(wrfhelper.settings['run_dir'], outfile) + " >> " + logfile)

        wrfhelper.write_simulated_flask(obs_id=obs_ids_write,
                                        simulated=ens_sim_write,
                                        nmembers=nmembers,
                                        outfile=outfile)

        os.chdir(cwd)
        os.system("echo 'Done' >> " + logfile)
```

wrfchem_flask_helper.py

```
1.  import os
2.  import shutil
3.  import re
4.  import glob
5.  import bisect
6.  import copy
7.  import numpy as np
8.  import netCDF4 as nc
9.  import datetime as dt
10. import wrf
11. import f90nml
12. import pickle
13.
14.
15. # CTDAS modules
16. import da.tools.io4 as io
17. from da.tools.wrfchem.utilities import utilities
18.
19.
20. class WRFChemHelper_flask(object):
21.     """Contains helper functions for sampling WRF-Chem"""
22.     def __init__(self, settings):
23.         self.settings = settings
24.
25.     def validate_settings(self, needed_items=[]):
26.         """
27.         This is based on WRFChemOO._validate_rc
28.         """
29.
30.         if len(needed_items)==0:
31.             return
32.
33.         for key in needed_items:
34.             if key not in self.settings:
35.                 msg = "Missing a required value in settings: %s" % key
36.                 raise IOError(msg)
37.
38.     @staticmethod
39.     def read_namelist(dirname):
40.         """Read run settings from namelist.input file in dirname"""
41.         nml_file = os.path.join(dirname, "namelist.input")
42.         namelist = f90nml.read(nml_file)
43.
44.         list_vars = ["e_we",
45.                      "e_sn",
46.                      "parent_id",
47.                      "parent_grid_ratio",
48.                      "i_parent_start",
49.                      "j_parent_start"
50.                     ]
51.         for v in list_vars:
52.             if not isinstance(namelist["domains"][v], list):
53.                 namelist["domains"][v] = [namelist["domains"][v]]
54.
55.         return namelist
56.
57.     def locate_domain(self, lat, lon, alt):   #Added altitude location - Ioanna
58.         """
59.         Input
60.         -----
61.         lat: Single values or lists or np.arrays
62.         lon: Single values or lists or np.arrays
63.
```

```
        Output
        ------
        - xy-coordinates in finest domain that contains the coordinates
        - finest domain
        -z
        """

        if not hasattr(self, "_loc_files"):
            raise RuntimeError("Must call open_wrf_loc_files first.")

        # Work with arrays internally.
        lat = np.array(lat, ndmin=1)
        lon = np.array(lon, ndmin=1)

        alt = np.array(alt, ndmin=1)

        # Get coordinates of the observation in all domains
        ndomains = self.namelist["domains"]["max_dom"]

        # Get domain sizes in xy on mass (=unstaggered) grid (hence the -1)
        dom_size_x    =    np.array(self.namelist['domains']['e_we'],    dtype=float, ndmin=1) - 1
        dom_size_y    =    np.array(self.namelist['domains']['e_sn'],    dtype=float, ndmin=1) - 1


        # Initialize  output
        xy = np.zeros((len(lat), 2))
        xy[:] = np.nan
        finest_domain = np.zeros((len(lat), ), int)

        z = np.zeros(len(alt))
        z[:] = np.nan

        # Since wrf.ll_to_xy takes very long, I save a bit of time here
        # by starting in the finest domain and processing only
        # observations that weren't previously found.
        for n in range(ndomains-1, -1, -1):
                    # Get xy for this domain
                    # Only process what you haven't processed
                    sel = np.where(finest_domain == 0)[0]

                    # In case all domains where set
                    if len(sel)==0:
                        break

                    x, y = wrf.ll_to_xy(wrfin=self._loc_files[n],
                                        latitude=lat[sel],
                                        longitude=lon[sel],
                                        meta=False,
                                        as_int=False)


                    # For each domain, check if the observation is inside the
                    # domain extent
                    # I put the edges on -0.5 and e_we/e_sn - 0.5.

                    # To be able to iterate over x and y in case they're scalars:
                    x = np.array(x, ndmin=1)
                    y = np.array(y, ndmin=1)

                    # Test: inside domain?
                    # The -1 here are because of 0-based indices, and the -/+ 0.5 are
                    # because that's halfway to the next mass-staggered point and
                    # I treat the WRF grid as boxes.
```

```
                    x_in = [(-0.5 <= x_) and (x_ <= dom_size_x[n] - 1 + 0.5) for x_
    in x]
                    y_in = [(-0.5 <= y_) and (y_ <= dom_size_y[n] - 1 + 0.5) for y_
    in y]

                    # Save domain, x and y at these locations
                    in_this_dom = np.where(np.logical_and(x_in, y_in))[0]
                    finest_domain[sel[in_this_dom]] = n + 1
                    # If domain = 1, set _all_ xy to see where they end up
                    if n==0:
                        xy[sel, 0] = x
                        xy[sel, 1] = y
                    else:
                        xy[sel[in_this_dom], 0] = x[in_this_dom]
                        xy[sel[in_this_dom], 1] = y[in_this_dom]


                    model_z_all_ll = wrf.getvar(wrfin=self._loc_files[n],
                                                varname= 'height_agl',
                                                timeidx=  0,
                                                units='m',
                                                squeeze=True,
                                                meta=False)

                    x = np.round(x).astype(int)
                    y = np.round(y).astype(int)

                    model_z = list()
                    z = list()

                    model_z = np.array([(model_z_all_ll[:, i,i]) for i, (x_,y_) in
    enumerate(zip(x, y))])
                    z = np.array([np.abs(model_z[i,:] - alt[sel[i]]).argmin() for i
    in range(len(x))])   #index of z

                # Return the indices and domain
                return xy, finest_domain, z

            def get_groups_space_time(self, dat, time_bins, only_in=False):
                """
                Returns a dictionary of lists of indices of observations in dat,
                where keys are tuples of (time bin, x bin, y bin and wrf
                domain), and values are indices of the observations that fall
                within this bin.
                """

                # Time groups
                id_t = np.array([bisect.bisect_right(time_bins, dat_date)
                                 for dat_date in dat['date']],
                                int)

                # Spatial groups (indices and domain)
                id_xy_f,     dom,     z     =     self.locate_domain(dat['latitude'],
    dat['longitude'], dat['altitude'])

                id_xy = np.round(id_xy_f).astype(int)



                # Version of only_in with sel: might be faster if sel is short -
                # I don't know! But the results for my test case where identical
                # (meaning 'domain' looked correct and identical) for both
                # options for only_in.

                # if only_in:
                #     # Yes, 0<id_t is correct: in bisect_right, it means the
```

```
188.                 #     # value is below the lowest sequence value.
189.                 #     time_in = np.logical_and(0<id_t, id_t<len(time_bins))
190.                 #     space_in = dom != 0
191.                 #     sel = np.where(np.logical_and(time_in, space_in))[0]
192.                 #
193.                 # else:
194.                 #     sel = range(len(id_t))
195.                 #
196.                 #  indices  =  get_index_groups(id_t[sel],  id_xy[sel,  0],  id_xy[sel,
    1], dom[sel])
197.                 #
198.                 # # Now have to account for sel again!
199.                 # if only_in:
200.                 #     for k, v in indices.iteritems():
201.                 #         indices[k] = sel[v]
202.
203.                 # Version of only_in  without sel: might be faster if sel is
204.                 # long - I don't know! But the results for my test case where
205.                 # identical (meaning 'domain' looked correct and identical)
206.
207.                 # Indices for all groups:
208.                 indices = utilities.get_index_groups(id_t, id_xy[:, 0], id_xy[:, 1],
    dom)
209.
210.                 # Throw the out-of-domain ones out here already
211.                 if only_in:
212.                     # Remove the index groups where domain is 0 (= outside of
213.                     # domain). Need to iterate over a list, because with an
214.                     # iterator, python complains that the dictionary changed
215.                     # size during iteration.
216.                     for k in list(indices.keys()):
217.                         if k[3] == 0:
218.                             del indices[k]
219.                     # Equivalent to the above 3 lines, don't know what's faster:
220.                     # groups = {k: v for k, v in groups.iteritems() if k[3] != 0}
221.
222.                 return indices
223.
224.
225.             @staticmethod
226.             def times_in_wrf_file(ncf):
227.                 """
228.                 Returns the times in netCDF4.Dataset ncf as datetime object
229.                 """
230.                 times_nc = ncf.variables["Times"]
231.                 times_chr = []
232.                 for nt in range(times_nc.shape[0]):
233.                     # freum 2021-07-11: with the migration to python3, need to
234.                     # replace the string - conversion. Hope "utf-8" works
235.                     # always.
236.                     # times_chr.append(times_nc[nt, :].tostring())
237.                     times_chr.append(str(times_nc[nt, :], "utf-8"))
238.
239.                 times_dtm = [dt.datetime.strptime(t_chr, "%Y-%m-%d_%H:%M:%S")
240.                              for t_chr in times_chr]
241.
242.                 return times_dtm
243.
244.             def wrf_times(self, file_list):
245.                 """Read all times in a list of wrf files
246.
247.                 Output
248.                 ------
249.                 - 1D-array containing all times
250.                 - 1D-array containing start indices of each file
251.                 """
```

```
252.
253.             times = list()
254.             start_indices = np.ndarray((len(file_list), ), int)
255.             for nf in range(len(file_list)):
256.                 ncf = nc.Dataset(file_list[nf])
257.                 times_this = self.times_in_wrf_file(ncf)
258.                 start_indices[nf] = len(times)
259.                 times += times_this
260.                 ncf.close()
261.
262.             return times, start_indices
263.
264.         def open_wrf_location_files(self):
265.             """
266.             Keep a description of a small wrf file for each domain in memory
267.             to locate observations.
268.             Appends _loc_file to self.
269.
270.             Note: Should be edited out of the code.
271.             """
272.
273.             ndomains = self.namelist["domains"]["max_dom"]
274.             path = self.settings["run_dir"]
275.             pattern = "wrfinput_d%02d"
276.             self._loc_files = list()
277.             for nd in range(1, ndomains+1):
278.                 fp = os.path.join(path, pattern % nd)
279.                 self._loc_files.append(nc.Dataset(fp, "r"))
280.
281.         def close_wrf_location_files(self):
282.             """See _open_wrf_location_files"""
283.             for loc_file in self._loc_files:
284.                 loc_file.close()
285.
286.         def wrf_filename_times(self, prefix):
287.             """Get timestamps in wrf file names in run directory."""
288.
289.             # List all filenames
290.             files = self.get_wrf_filenames(prefix + "*")
291.             # Only use d01 files, pattern should be the same for all domains
292.             pattern = os.path.join(self.settings["run_dir"], prefix)
293.             files = [f for f in files if re.search(pattern, f)]
294.             # Extract timestamp from filename
295.             # Format is %Y-%m-%d_%H:%M:%S at the end of the filename
296.             pattern_time = "%Y-%m-%d_%H:%M:%S"
297.             len_tstamp = len(pattern_time) + 2
298.             times = [dt.datetime.strptime(f[-len_tstamp:], pattern_time)
299.                      for f in files]
300.
301.             return times
302.
303.         def get_wrf_filenames(self, glob_pattern):
304.             """
305.             Gets the filenames in self.settings["run_dir"] that follow
306.             glob_pattern, excluding those that end with
307.             self.settings["original_save_suffix"]
308.             """
309.             path = self.settings["run_dir"]
310.             # All files...
311.             wfiles = glob.glob(os.path.join(path, glob_pattern))
312.             # All originals
313.             orig_suf = self.settings["original_save_suffix"]
314.             opattern = glob_pattern + orig_suf
315.             ofiles = glob.glob(os.path.join(path, opattern))
316.
317.             # All files except all originals:
```

```
318.                files = [x for x in wfiles if x not in ofiles]
319.
320.                # I need this sorted too often to not do it here.
321.                files = np.sort(files).tolist()
322.                return files
323.
324.
325.            def sample_flask(self, dat, loc, fields_list):
326.
327.                # Initialize output
328.                tc    =    np.ndarray(shape=(len(dat["obs_num"]),    len(fields_list)),
    dtype=float)
329.                tc[:] = float("nan")
330.
331.                # Process by domain
332.                UD = list(set(loc["domain"]))
333.                #for dom in UD[1:]:
334.                for dom in UD:
335.                    idd = np.nonzero(loc["domain"] == dom)[0]
336.                    # Process by id_t
337.                    UT = list(set(loc["id_t"][idd]))
338.                    for time_id in UT:
339.                        # Coordinates to process
340.                        idt = idd[np.nonzero(loc["id_t"][idd] == time_id)[0]]
341.                        # Get tracer ensemble profiles
342.                        profiles = self._read_and_intrp_v(loc, fields_list, time_id,
    idt)
343.                # Here it starts to make sense to loop over individual observations
344.                        for nidt in range(len(idt)):
345.                            nobs = idt[nidt]
346.                           # Compute flasks
347.                            for nf in range(len(fields_list)):
348.
349.                                # Model retrieval
350.                                tc[nobs, nf] = profiles[nf][nidt]
351.
352.                return tc
353.
354.            @staticmethod
355.            def _read_and_intrp_v(loc, fields_list, time_id, idp):
356.                """
357.                Helper function for sample_flasks.
358.                read_and_intrp, but vectorized.
359.                Reads in fields and interpolates
360.                them linearly in time.
361.
362.                Output
363.                ------
364.                List of temporally interpolated fields, one entry per member of
365.                fields_list.
366.                """
367.
368.                var_intrp_l = list()
369.
370.                # Check we were really called with observations for just one domain
371.                domains = set(loc["domain"][idp])
372.                if len(domains) > 1:
373.                    raise  ValueError("I  can  only  operate  on  idp  with  identical
    domains.")
374.                dom = domains.pop()
375.
376.                # Select input files
377.                id_file0  =  bisect.bisect_right(loc["file_start_time_indices"][dom],
    time_id) - 1
378.                id_file1  =  bisect.bisect_right(loc["file_start_time_indices"][dom],
    time_id+1) - 1
```

```
379.                 if id_file0 < 0 or id_file1 < 0:
380.                     raise ValueError("This shouldn't happen.")
381.
382.                 # Get time id in file
383.                 id_t_file0 = time_id - loc["file_start_time_indices"][dom][id_file0]
384.                 id_t_file1                  =                time_id+1                -
    loc["file_start_time_indices"][dom][id_file1]
385.
386.                 # Open files
387.                 nc0 = nc.Dataset(loc["files"][dom][id_file0], "r")
388.                 nc1 = nc.Dataset(loc["files"][dom][id_file1], "r")
389.                 # Per field to sample
390.                 for field in fields_list:
391.                     # Read input file
392.                     field0 = wrf.getvar(wrfin=nc0,
393.                                         varname=field,
394.                                         timeidx=id_t_file0,
395.                                         squeeze=False,
396.                                         meta=False)
397.
398.                     field1 = wrf.getvar(wrfin=nc1,
399.                                         varname=field,
400.                                         timeidx=id_t_file1,
401.                                         squeeze=False,
402.                                         meta=False)
403.
404.                     if len(field0.shape) == 4:
405.                         # Sample field at timesteps before and after observation
406.                         # They are ordered nt x nz x ny x nx
407.                         # var0 will have shape (len(idp),len(profile))
408.                         var0   =   field0[0,   loc["z"][idp],   loc["id_xy"][idp,   1],
    loc["id_xy"][idp, 0]]
409.                         var1   =   field1[0,   loc["z"][idp],   loc["id_xy"][idp,   1],
    loc["id_xy"][idp, 0]]
410.                         # Repeat frac_t for profile size
411.                         frac_t_   =   np.array(loc["frac_t"][idp])#.reshape((len(idp),
    1)).repeat(var0.shape[1], 1)
412.                     elif len(field0.shape) == 3:
413.                         # var0 will have shape (len(idp),)
414.                         var0 = field0[0, loc["id_xy"][idp, 1], loc["id_xy"][idp, 0]]
415.                         var1 = field1[0, loc["id_xy"][idp, 1], loc["id_xy"][idp, 0]]
416.                         frac_t_ = np.array(loc["frac_t"][idp])
417.                     elif len(field0.shape) == 2:
418.                         # var0 will have shape (len(idp),len(profile))
419.                         # This is for ZNW, which is saved as (time_coordinate,
420.                         # vertical_coordinate)
421.                         var0 = field0[[0]*len(idp), :]
422.                         var1 = field1[[0]*len(idp), :]
423.                         frac_t_   =   np.array(loc["frac_t"][idp]).reshape((len(idp),
    1)).repeat(var0.shape[1], 1)
424.                     else:
425.                         raise ValueError("Can't deal with field with %d dimensions."
    % len(field0.shape))
426.
427.                     # Interpolate in time
428.                     var_intrp_l.append(var0*frac_t_ + var1*(1. - frac_t_))
429.
430.                 nc0.close()
431.                 nc1.close()
432.
433.                 return var_intrp_l
434.
435.             @staticmethod
436.             def read_sampling_coords(sampling_coords_file, id0=None, id1=None):
437.                 """Read in samples"""
438.
```

```
439.                ncf = nc.Dataset(sampling_coords_file, "r")
440.                if id0 is None:
441.                    id0 = 0
442.                if id1 is None:
443.                    id1 = len(ncf.dimensions['obs'])
444.
445.                dat = dict(
446.                    obs_num=np.array(ncf.variables["obs_num"][id0:id1]),
447.                    date=ncf.variables["date_components"][id0:id1],
448.                    latitude=np.array(ncf.variables["latitude"][id0:id1]),
449.                    longitude=np.array(ncf.variables["longitude"][id0:id1]),
450.                    altitude=np.array(ncf.variables["altitude"][id0:id1]),
451.                    #strategy=np.array(ncf.variables["sampling_stategy"][id0:id1]),
452.                    #evn=np.array(ncf.variables["obs_id"][id0:id1]),
453.                    obs=np.array(ncf.variables["observed"][id0:id1]),
454.                    mdm=np.array(ncf.variables["modeldatamismatch"][id0:id1])
455.                    )
456.
457.                ncf.close()
458.
459.                # Convert date to datetime object
460.                dat["time"] = [dt.datetime(*x) for x in dat["date"]]
461.
462.                return dat
463.
464.            @staticmethod
465.            def write_simulated_flask(obs_id, simulated, nmembers, outfile):
466.                """Write simulated observations to file."""
467.
468.                # Output format: see obs_xco2_fr
469.
470.                f = io.CT_CDF(outfile, method="create")
471.
472.                dimid = f.createDimension("obs_num", size=None)
473.                dimid = ("obs_num",)
474.                savedict = io.std_savedict.copy()
475.                savedict["name"] = "obs_num"
476.                savedict["dtype"] = "int64"
477.                savedict["long_name"] = "Unique_Dataset_observation_index_number"
478.                savedict["units"] = ""
479.                savedict["dims"] = dimid
480.                savedict["comment"] = "Format as in input"
481.                savedict["values"] = obs_id.tolist()
482.                f.add_data(savedict, nsets=0)
483.
484.                dimmember = f.createDimension("nmembers", size=nmembers)
485.                dimmember = ("nmembers",)
486.                savedict = io.std_savedict.copy()
487.                savedict["name"] = "flask"
488.                savedict["dtype"] = "float"
489.                savedict["long_name"] = "Simulated flask"
490.                savedict["units"] = "??"
491.                savedict["dims"] = dimid + dimmember
492.                savedict["comment"] = "Simulated model value created by WRFChemOO"
493.                savedict["values"] = simulated.tolist()
494.                f.add_data(savedict, nsets=0)
495.
496.                f.close()
497.
498.            @staticmethod
499.            def save_file_with_timestamp(file_path, out_dir, suffix=""):
500.                """ Saves a file to with a timestamp"""
501.                nowstamp = dt.datetime.now().strftime("_%Y-%m-%d_%H:%M:%S")
502.                new_name = os.path.basename(file_path) + suffix + nowstamp
503.                new_path = os.path.join(out_dir, new_name)
504.                shutil.copy2(file_path, new_path)
```

```
505.
506.
507.        if __name__ == "__main__":
508.            pass
```

REFERENCES

https://www2.acom.ucar.edu/wrf-chem

https://www.iup.uni-bremen.de/carbon_ghg/products/tropomi_wfmd/index_v12.php

https://www.ecmwf.int/en/research/data-assimilation

https://finokalia.chemistry.uoc.gr/

https://www2.acom.ucar.edu/modeling/finn-fire-inventory-ncar

https://ads.atmosphere.copernicus.eu/cdsapp#!/dataset/cams-global-emission-inventories?tab=overview

Ahmadov, R., Gerbig, C., Kretschmer, R., Körner, S., Rödenbeck, C., Bousquet, P., & Ramonet, M. (2009). Comparing high resolution WRF-VPRM simulations and two global CO 2 transport models with coastal tower measurements of CO2. *Biogeosciences*, *6*(5), 807–817. https://doi.org/10.5194/bg-6-807-2009

Atkinson, R., Baulch, D. L., Cox, R. A., Crowley, J. N., Hampson, R. F., Hynes, R. G., Jenkin, M. E., Rossi, M. J., & Troe, J. (2006). Evaluated kinetic and photochemical data for atmospheric chemistry: Volume II – gas phase reactions of organic species. *Atmospheric Chemistry and Physics*, *6*(11), 3625–4055. https://doi.org/10.5194/acp-6-3625-2006

Beck, V., Koch, T., Kretschmer, R., Marshall, J., Ahmadov, R., Gerbig, C., Pillai, D., & Heimann, M. (2011). *The WRF Greenhouse Gas Model (WRF-GHG) Technical Report*. https://www.bgc-jena.mpg.de/bgc-systems/pmwiki2/uploads/Download/Wrf-ghg/WRF-GHG_Techn_Report.pdf

Bogner, J., & Spokas, K. (1993). Landfill CH4: Rates, fates, and role in global carbon cycle. *Chemosphere*, *26*(1–4), 369–386. https://doi.org/10.1016/0045-6535(93)90432-5

Brasseur, G. P., & Jacob, D. J. (2017). Modeling of atmospheric chemistry. *Modeling of Atmospheric Chemistry*, *March*, 1–606. https://doi.org/10.1017/9781316544754

Bruhwiler, L., Dlugokencky, E., Masarie, K., Ishizawa, M., Andrews, A., Miller, J., Sweeney, C., Tans, P., & Worthy, D. (2014). CarbonTracker-CH4: An assimilation system for estimating emissions of atmospheric methane. *Atmospheric Chemistry and Physics*, *14*(16), 8269–8293. https://doi.org/10.5194/acp-14-8269-2014

Bruhwiler, L. M. P., Michalak, A. M., Peters, W., Baker, D. F., & Tans, P. (2005). An improved Kalman smoother for atmospheric inversions. *Atmospheric Chemistry and Physics*, *5*(10), 2691–2702. https://doi.org/10.5194/acp-5-2691-2005

Chang, J., Peng, S., Ciais, P., Saunois, M., Dangal, S. R. S., Herrero, M., Havlík, P., Tian, H., & Bousquet, P. (2019). Revisiting enteric methane emissions from domestic ruminants and their δ13CCH4 source signature. *Nature Communications*, *10*(1), 1–14. https://doi.org/10.1038/s41467-019-11066-3

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

Field Code Changed

Gourdji, S. M., Yadav, V., Karion, A., Mueller, K. L., Conley, S., Ryerson, T., Nehrkorn, T., & Kort, E. A. (2018). Reducing errors in aircraft atmospheric inversion estimates of point-source emissions: The Aliso Canyon natural gas leak as a natural tracer experiment. *Environmental Research Letters*, *13*(4). https://doi.org/10.1088/1748-9326/aab049

Grell, G. A., Peckham, S. E., Schmitz, R., McKeen, S. A., Frost, G., Skamarock, W. C., & Eder, B. (2005). Fully coupled "online" chemistry within the WRF model. *Atmospheric Environment*, *39*(37), 6957–6975. https://doi.org/10.1016/j.atmosenv.2005.04.027

Hausmann, P., Sussmann, R., & Smale, D. (2016). Contribution of oil and natural gas production to renewed increase in atmospheric methane (2007-2014): Top-down estimate from ethane and methane column observations. *Atmospheric Chemistry and Physics*, *16*(5), 3227–3244. https://doi.org/10.5194/acp-16-3227-2016

Hersbach, H., Bell, B., Berrisford, P., Hirahara, S., Horányi, A., Muñoz-Sabater, J., Nicolas, J., Peubey, C., Radu, R., Schepers, D., Simmons, A., Soci, C., Abdalla, S., Abellan, X., Balsamo, G., Bechtold, P., Biavati, G., Bidlot, J., Bonavita, M., … Thépaut, J. N. (2020). The ERA5 global reanalysis. *Quarterly Journal of the Royal Meteorological Society*, *146*(730), 1999–2049. https://doi.org/10.1002/qj.3803

Huijnen, V., Williams, J., Van Weele, M., Van Noije, T., Krol, M., Dentener, F., Segers, A., Houweling, S., Peters, W., De Laat, J., Boersma, F., Bergamaschi, P., Van Velthoven, P., Le Sager, P., Eskes, H., Alkemade, F., Scheele, R., Nédélec, P., & Pätz, H. W. (2010). The global chemistry transport model TM5: Description and evaluation of the tropospheric chemistry version 3.0. *Geoscientific Model Development*, *3*(2), 445–473. https://doi.org/10.5194/gmd-3-445-2010

Kaplan, J. O. (2002). Wetlands at the Last Glacial Maximum: Distribution and methane emissions. *Geophysical Research Letters*, *29*(6), 3–6. https://doi.org/10.1029/2001GL013366

Kolb, S., & Horn, M. A. (2012). Microbial CH4 and N2O consumption in acidic wetlands. *Frontiers in Microbiology*, *3*(MAR). https://doi.org/10.3389/fmicb.2012.00078

Levine, J. S. (2000). Biomass Burning and the Production of Greenhouse Gases. *Climate Biosphere Interaction: Biogenic Emissions and Environmental Effects of Climate Change*, 1–13.

Miller, S. M., Michalak, A. M., & Levi, P. J. (2014). Atmospheric inverse modeling with known physical bounds: An example from trace gas emissions. *Geoscientific Model Development*, *7*(1), 303–315. https://doi.org/10.5194/gmd-7-303-2014

Ministry of Environment and Energy. (2021). *NATIONAL INVENTORY REPORT OF GREECE FOR GREENHOUSE AND OTHER GASES FOR THE YEARS 1990-2019*. *March*.

Müller, J. F., Liu, Z., Nguyen, V. S., Stavrakou, T., Harvey, J. N., & Peeters, J. (2016). The reaction of methyl peroxy and hydroxyl radicals as a major source of atmospheric methanol. *Nature Communications*, *7*(May), 1–11. https://doi.org/10.1038/ncomms13213

Patra, A., Park, T., Kim, M., & Yu, Z. (2017). Rumen methanogens and mitigation of methane emission by anti-methanogenic compounds and substances. *Journal of Animal Science and Biotechnology*, *8*(1), 1–18. https://doi.org/10.1186/s40104-017-0145-9

Paudel, R., Mahowald, N. M., Hess, P. G. M., Meng, L., & Riley, W. J. (2016). Attribution of changes in global wetland methane emissions from pre-industrial to present using CLM4.5-BGC. *Environmental Research Letters*, *11*(3), 34020. https://doi.org/10.1088/1748-9326/11/3/034020

Peters, W., Miller, J. B., Whitaker, J., Denning, A. S., Hirsch, A., Krol, M. C., Zupanski, D., Bruhwiler, L., & Tans, P. P. (2005). An ensemble data assimilation system to estimate CO2 surface fluxes from atmospheric trace gas observations. *Journal of Geophysical Research Atmospheres*, *110*(24), 1–18. https://doi.org/10.1029/2005JD006157

Poulter, B., Bousquet, P., Canadell, J. G., Ciais, P., Peregon, A., Saunois, M., Arora, V. K., Beerling, D. J., Brovkin, V., Jones, C. D., Joos, F., Gedney, N., Ito, A., Kleinen, T., Koven, C. D., McDonald, K., Melton, J. R., Peng, C., Peng, S., … Zhu, Q. (2017). Global wetland contribution to 2000-2012 atmospheric methane growth rate dynamics. *Environmental Research Letters*, *12*(9). https://doi.org/10.1088/1748-9326/aa8391

Ridgwell, J., Marshall, J., & Gregson, K. (1999). Consumption of atmospheric methane by soils: A process-based model. *Global Biogeochemical Cycles*, *13*(1), 59–70.

Ruppel, C. D., & Kessler, J. D. (2017). The interaction of climate change and methane hydrates. *Reviews of Geophysics*, *55*(1), 126–168. https://doi.org/10.1002/2016RG000534

Salawitch, R. J., Canty, T. P., Hope, A. P., Tribett, W. R., & Bennett, B. F. (n.d.). *Paris Climate Agreement : beacon of hope*.

Sanchis, E., Ferrer, M., Torres, A. G., Cambra-López, M., & Calvet, S. (2012). Effect of water and straw management practices on methane emissions from rice fields: A review through a meta-analysis. *Environmental Engineering Science*, *29*(12), 1053–1062. https://doi.org/10.1089/ees.2012.0006

Sanderson, M. S. (1996). Biomass of termites and their emissions of methane and carbon dioxide: A global database. *Global Biogeochemical Cycles*, *10*(4), 543–557. https://doi.org/10.1029/96GB01893

Schneising, O., Buchwitz, M., Reuter, M., Bovensmann, H., Burrows, J. P., Borsdorff, T., Deutscher, N. M., Feist, D. G., Griffith, D. W. T., Hase, F., Hermans, C., Iraci, L. T., Kivi, R., Landgraf, J., Morino, I., Notholt, J., Petri, C., Pollard, D. F., Roche, S., … Wunch, D. (2019). A scientific algorithm to simultaneously retrieve carbon monoxide and methane from TROPOMI onboard Sentinel-5 Precursor. *Atmospheric Measurement Techniques*, *12*(12), 6771–6802. https://doi.org/10.5194/amt-12-6771-2019

Seinfeld, J. H., Pandis, S. N., & Noone, K. (1998). Atmospheric Chemistry and Physics: From Air Pollution to Climate Change . In *Physics Today* (Vol. 51, Issue 10). https://doi.org/10.1063/1.882420

Sitch, S., Smith, B., Prentice, I. C., Arneth, A., Bondeau, A., Cramer, W., Kaplan, J. O., Levis, S., Lucht, W., Sykes, M. T., Thonicke, K., & Venevsky, S. (2003).

Evaluation of ecosystem dynamics, plant geography and terrestrial carbon cycling in the LPJ dynamic global vegetation model. *Global Change Biology*, *9*(2), 161–185. https://doi.org/10.1046/j.1365-2486.2003.00569.x

Skamarock, W. C., Klemp, J. B., Dudhia, J. B., Gill, D. O., Barker, D. M., Duda, M. G., Huang, X.-Y., Wang, W., & Powers, J. G. (2008). A description of the Advanced Research WRF Version 3, NCAR Technical Note TN-475+STR. *Technical Report*, *June*, 113.

Van Der Laan-Luijkx, I. T., Van Der Velde, I. R., Van Der Veen, E., Tsuruta, A., Stanislawska, K., Babenhauserheide, A., Fang Zhang, H., Liu, Y., He, W., Chen, H., Masarie, K. A., Krol, M. C., & Peters, W. (2017). The CarbonTracker Data Assimilation Shell (CTDAS) v1.0: Implementation and global carbon balance 2001-2015. *Geoscientific Model Development*, *10*(7), 2785–2800. https://doi.org/10.5194/gmd-10-2785-2017

Whitaker, J. S., & Hamill, T. M. (2002). Ensemble data assimilation without perturbed observations. *Monthly Weather Review*, *130*(7), 1913–1924. https://doi.org/10.1175/1520-0493(2002)130<1913:EDAWPO>2.0.CO;2

Zhang, Z., Fluet-Chouinard, E., Jensen, K., McDonald, K., Hugelius, G., Gumbricht, T., Carroll, M., Prigent, C., Bartsch, A., & Poulter, B. (2021). Development of the global dataset of Wetland Area and Dynamics for Methane Modeling (WAD2M). *Earth System Science Data*, *13*(5), 2001–2023. https://doi.org/10.5194/essd-13-2001-2021

Zhao, H., Themelis, N. J., Bourtsalas, A., & McGillis, W. R. (2019). Methane Emissions from Landfills. *ResearchGate*, *May*, 1–98.

Zhou, M., Langerock, B., Vigouroux, C., Sha, M. K., Ramonet, M., Delmotte, M., Mahieu, E., Bader, W., Hermans, C., Kumps, N., Metzger, J. M., Duflot, V., Wang, Z., Palm, M., & De Mazière, M. (2018). Atmospheric CO and CH4 time series and seasonal variations on Reunion Island from ground-based in situ and FTIR (NDACC and TCCON) measurements. *Atmospheric Chemistry and Physics*, *18*(19), 13881–13901. https://doi.org/10.5194/acp-18-13881-2018