FLOW: First Learn tO Walk

Karina Ramos Musalem, CICESE
Noah Rosenberg, University of Washington
Shreyas Gaikwad, University of Texas at Austin

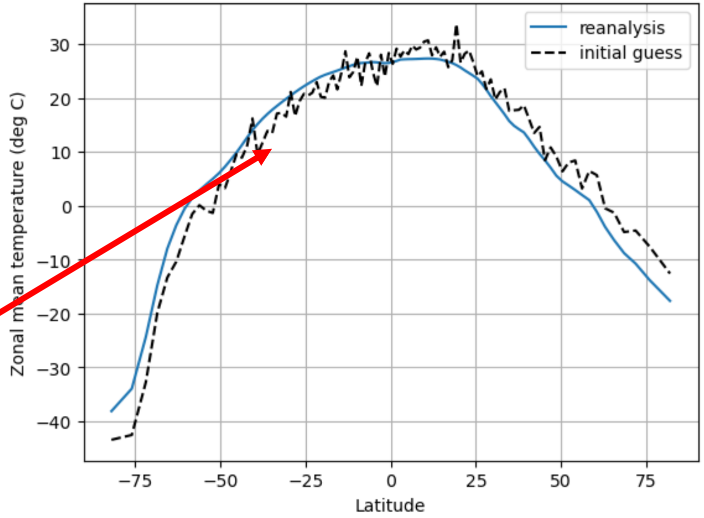
Question: Using a simple climate model and its AD-generated adjoint, can we perform a state estimation and recover realistic control parameters?

The Budyko-Sellers Model: A review from Day 3

tendency — Insolation — OLR — Downgradient diffusion

$$C(\phi)\frac{\partial T_s}{\partial t} = [1-\alpha(T_s)]\,Q(\phi) - \epsilon\sigma\,[\cancel{\alpha}(T_s)]^4 + \frac{D}{\cos(\phi)}\frac{\partial}{\partial\phi}\left(\cos(\phi)\frac{\partial T_s}{\partial\phi}\right)$$

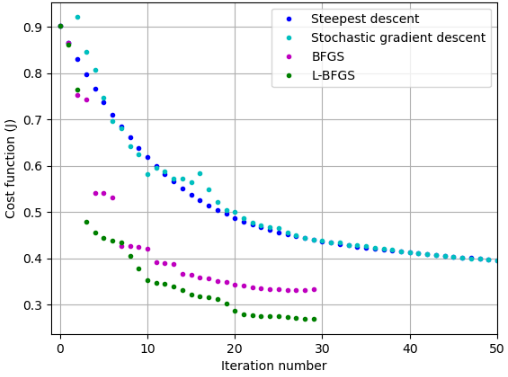
- 3 tunable parameters: Diffusivity D, emissivity ϵ, albedo α
- 1 state variable: temperature T_s
- Converges to steady state representing zonal-average, time-mean temperature



Can we push the initial guess towards the target using the adjoint gradient to various controls?

Set up as an optimization problem: minimize $J = (\Sigma\,(T_i - T_{i,Target})^2)^{0.5}/N$

Exploration 1: Implement and compare various gradient descent algorithms



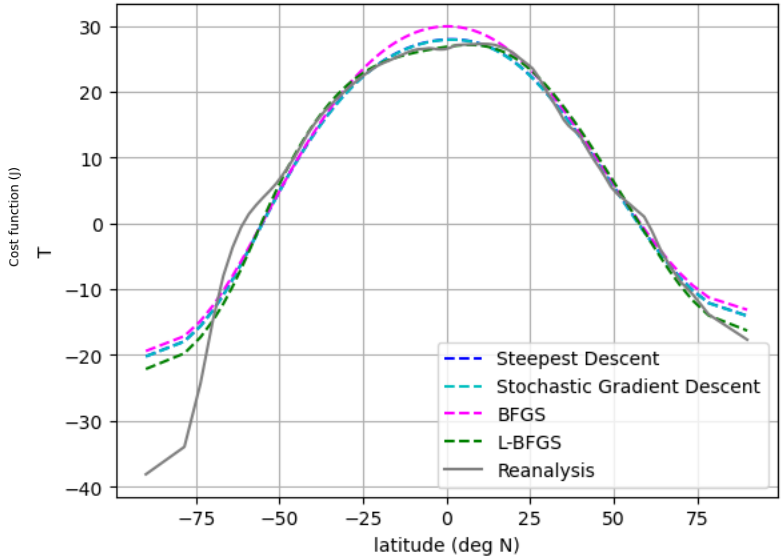
Steepest Descent: For a control vector $\mathbf{X}_i$ and gradient $\mathbf{g}_i$, let $\mathbf{X}_{i+1} = \mathbf{X}_i - a\mathbf{g}_i$ using some ~~learning~~ rate a

Newton: Let $\mathbf{X}_{i+1} = \mathbf{X}_i - a(\mathcal{H})^{-1}\mathbf{g}$, where $\mathcal{H}$ is the Hessian of J with respect to $\mathbf{X}$

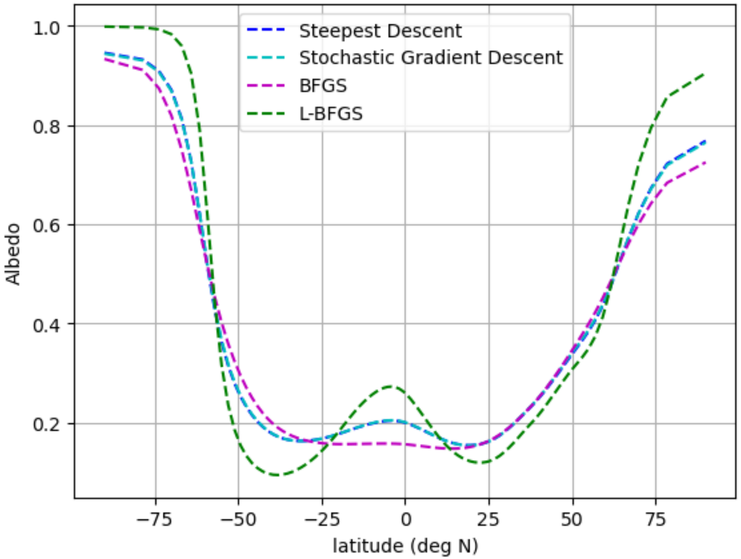
Quasi-Newton: Approximate $\mathbf{B} = (\mathcal{H})^{-1}$ using cheaper resources (BFGS: store full approximate B, L-BFGS: calculate from last *k* control and gradient vectors

Tired of waiting for your gradient descent to converge? Try using quasi-Newton methods!

Exploration 1: Implement and compare various gradient descent algorithms



Desc
= X
Let X
espe
wtor
(BFG
from



adient
essian
per
:
s

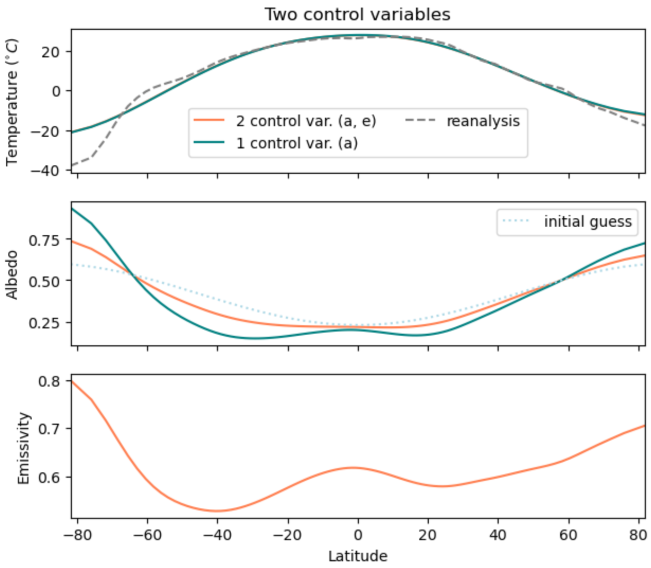
Tired of waiting for your gradient descent to converge? Try using quasi-Newton methods!

Exploration 2: How does the estimate change as we add and normalize controls?

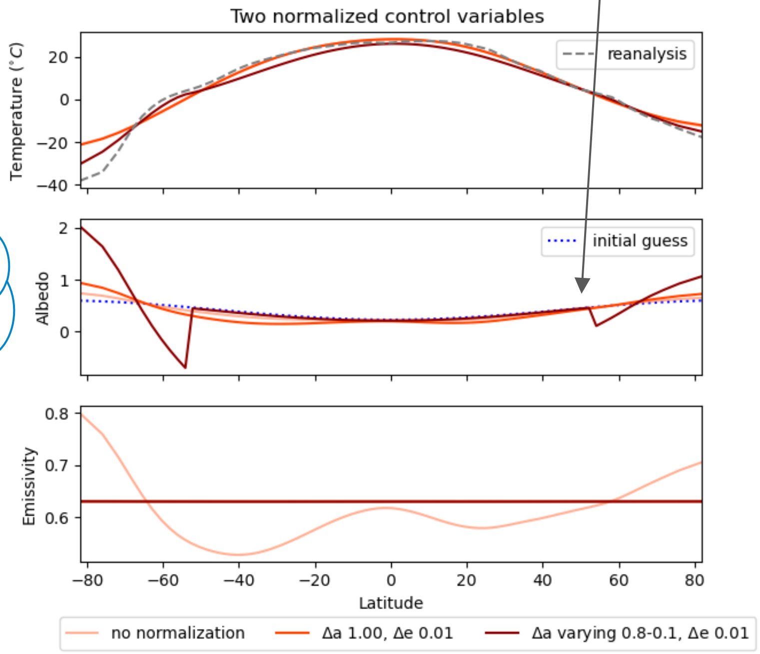
Albedo (a) + emissivity (e)
Initial guess a: 2nd Legendre polynomial, e : constant

Uncertainties for a and e have different in magnitudes and vary spatially (Eg. Δa with latitude)

Sensitive to choice of descent step and threshold for cost function!



But I have a perfectly constant albedo of O



FLOW
First Learn tO Walk

ECCO SUMMER SCHOOL

budyko_sellers_state_estimation.ipynb

State Estimation in the Budyko-Sellers energy balance model using algorithmic differentiation in JAX

This notebook runs the Budyko-Sellers 1-D energy balance model and uses its gradient to reduce the cost function J, which represents the mismatch between zonal mean temperature from NCEP and the model.

References:

Notes on the Budyko-Sellers Model

Notes on L-BFGS implementation

Acknowledgements to Shreyas Gaikwad and Ian Fenty for providing the Fortran code which was adapted for this notebook.

In [ ]:
```
!pip install jax jaxlib
```

In [2]:
```
import jax.numpy as jnp
from jax import grad

import numpy as np
import jax.numpy as jnp
from jax import grad, lax,jit
import sys
N = 100
num_controls = 1
import netCDF4 as nc
import matplotlib.pyplot as plt
from collections import deque
```

Step 0: download and plot the target data

We will use as our target the NCEP reanalysis surface temperature, averaged zonally and in time, to evolve our steady state temperature to.

In [8]:
```
## Import NCEP temperature, get zonal mean and extrapolate to grid size

XEDGES = jnp.linspace(-1.0, 1.0, N + 1)
X = 0.5 * (XEDGES[:-1] + XEDGES[1:])
LAT = jnp.arcsin(X) * 180.0 / jnp.pi
DX = X[1] - X[0]
```

budyko_sellers_controls.ipynb

State Estimation in the Budyko-Sellers energy balance model: Exploring the role of controls

This notebook runs the Budyko-Sellers 1-D energy balance model and uses its gradient to reduce the cost function J, which represents the mismatch between zonal mean temperature from NCEP and the model. It uses algorithmic differentiation in JAX.

We explore how the state estimation changes when we modify the number of controls and the uncertainties associated to them.

- We will solve for 1 contol (albedo) and 2 controls (albedo and emissivity).
- Explore how non-dimensionalizing the inital controls helps the assimilation and provides final control values that agree better with physics and expected values.

References:

Notes on the Budyko-Sellers Modelby Brian E. J. Rose (University at Albany)

Shreyas Gaikwad and Ian Fenty provided the Fortran code which was adapted for this notebook.

In [1]:
```
!pip install jax jaxlib
```

```
Collecting jax
  Using cached jax-0.6.1-py3-none-any.whl.metadata (13 kB)
Collecting jaxlib
  Using cached jaxlib-0.6.1-cp312-cp312-manylinux2014_x86_64.whl.metadata (1.2 kB)
Collecting ml_dtypes>=0.5.0 (from jax)
  Using cached ml_dtypes-0.5.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (21 kB)
Requirement already satisfied: numpy>=1.25 in /srv/conda/envs/notebook/lib/python3.12/site-packages (from jax)
(2.0.2)
Collecting opt_einsum (from jax)
  Using cached opt_einsum-3.4.0-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: scipy>=1.11.1 in /srv/conda/envs/notebook/lib/python3.12/site-packages (from jax)
(1.15.1)
Using cached jax-0.6.1-py3-none-any.whl (2.4 MB)
Using cached jaxlib-0.6.1-cp312-cp312-manylinux2014_x86_64.whl (89.1 MB)
Using cached ml_dtypes-0.5.1-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.7 MB)
Using cached opt_einsum-3.4.0-py3-none-any.whl (71 kB)
Installing collected packages: opt_einsum, ml_dtypes, jaxlib, jax
Successfully installed jax-0.6.1 jaxlib-0.6.1 ml_dtypes-0.5.1 opt_einsum-3.4.0
```

Product: two tutorial notebooks with self-contained state estimation procedures, easily expandable

Reflections

- State estimation is hard! Even with 100-300 controls, convergence was sensitive to initial guess, descent rate, algorithm used, constraints on controls, convergence conditions, and more.
- AD is also sensitive to data structures and other considerations in the cost function definition
- FORTRAN 77
- Gained skills in AD (jax + tapenade), fortran, gradient descent, constraints on parameters

What would we do with 1 more week?

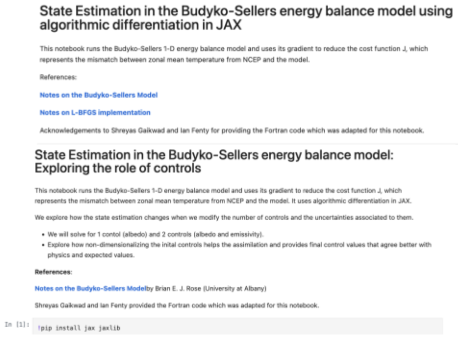
- How can we constrain uncertainties while also constraining the values of controls?
- Explore time dependence and hysteresis in the forward model–does this break our estimation?
- More complexity, more controls, more targets! (moisture?)

FLOW: First Learn tO Walk

Karina Ramos Musalem, CICESE
Noah Rosenberg, University of Washington

Question: Using a simple climate model and its AD-generated adjoint, can we perform a state estimation and recover realistic control parameters?

Product: two tutorial notebooks with self-contained state estimation procedures



State Estimation in the Budyko-Sellers energy balance model using algorithmic differentiation in JAX

This notebook runs the Budyko-Sellers 1-D energy balance model and uses its gradient to reduce the cost function J, which represents the mismatch between zonal mean temperature from NCEP and the model.

References:

Notes on the Budyko-Sellers Model

Notes on L-BFGS implementation

Acknowledgements to Shreyas Gaikwad and Ian Fenty for providing the Fortran code which was adapted for this notebook.

State Estimation in the Budyko-Sellers energy balance model: Exploring the role of controls

This notebook runs the Budyko-Sellers 1-D energy balance model and uses its gradient to reduce the cost function J, which represents the mismatch between zonal mean temperature from NCEP and the model. It uses algorithmic differentiation in JAX.

We explore how the state estimation changes when we modify the number of controls and the uncertainties associated to them.

- We will solve for 1 contol (albedo) and 2 controls (albedo and emissivity).
- Explore how non-dimensionalizing the inital controls helps the assimilation and provides final control values that agree better with physics and expected values.

References:

Notes on the Budyko-Sellers Modelby Brian E. J. Rose (University at Albany)

Shreyas Gaikwad and Ian Fenty provided the Fortran code which was adapted for this notebook.

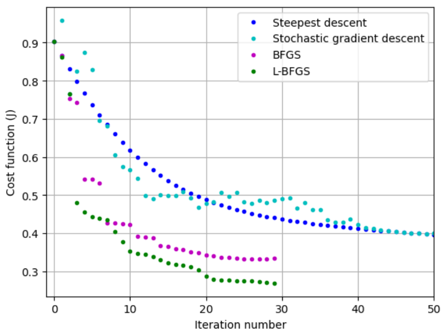
In [1]:
```
!pip install jax jaxlib
```

Reflections

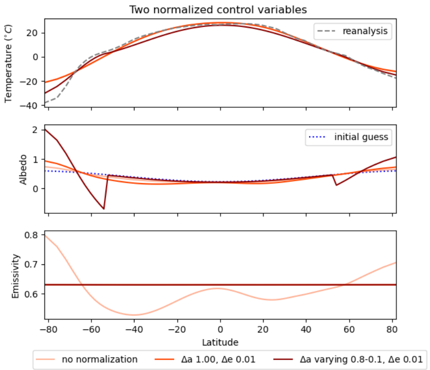
- State estimation is hard! Even with 100-300 controls, convergence was sensitive to initial guess, descent rate, algorithm used, constraints on controls, convergence conditions, and more.
- AD is also sensitive to data structures and other considerations in the cost function definition
- Gained skills in AD (jax + tapenade), fortran, gradient descent, constraints on parameters

tendency — Insolation — OLR — Downgradient diffusion

$$C(\phi)\frac{\partial T_s}{\partial t} = \left[1-\alpha(T_s)\right]Q(\phi) - \epsilon\sigma\left[\alpha(T_s)\right]^4 + \frac{D}{\cos(\phi)}\frac{\partial}{\partial\phi}\left(\cos(\phi)\frac{\partial T_s}{\partial\phi}\right)$$



Exploration 1: Implementation of various gradient descent algorithms



Exploration 2: How does the estimate change as we add and normalize controls?