"Off-Offline" Testing of Your Model

How and Why You Should Do It

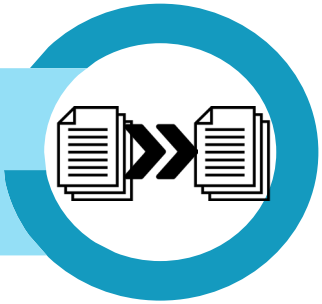
Adrianna Foster
Project Scientist II, NCAR CGD Terrestrial Sciences Section

NSF NCAR Land Model/Biogeochemistry 2024 Winter Working Group Meeting
Tuesday, February 25

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

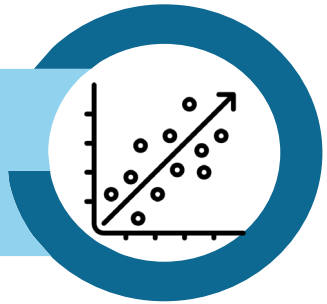
How do we normally test our models?

Regression/System Testing

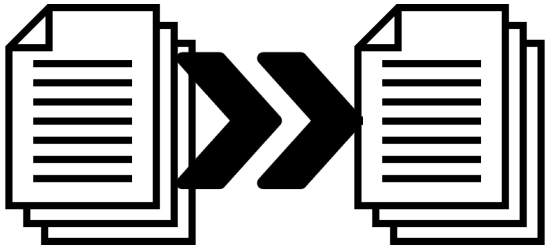


Unit Testing

Validation/Offline Testing



Regression & System Testing



- Ensures model produces consistent results
- Does the model produce the same results (i.e. “bit -for-bit”) when we expect it to?
  - restarts
  - different processors
  - updates that shouldn’t “change answers”

Your Software Engineer colleagues do these **ALL THE TIME**

No science involved – mostly ensures code is working as expected

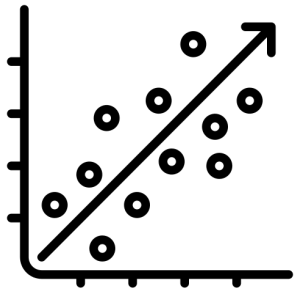
Unit Testing

- Tests “units” of software
- Well-defined inputs and outputs
  - checking for errors
  - generally only test one or a few methods at a time
  - must have PASS/FAIL condition

Your Software Engineer colleagues do these **FREQUENTLY** (for infrastructure code)

Still mostly ensuring software is working as expected

Model Validation



- Comparing production -run model output to observations
  - maybe with controlled conditions
- Crucial for verifying a model accurately represents reality
- Model calibration
- CLM – generally ”offline runs”

As Scientists this is what we do **ALL THE TIME**

Critical yet can be difficult with interactive model processes

“Off-offline” testing

- Test small, isolated pieces of the model
- Validate internal model behavior before running full-scale runs

```
You, 10 months ago | 1 author (You)
program FatesTestQuadSolvers

  use FatesConstantsMod, only : r8 => fates_r8
  use FatesUtilsMod,     only : QuadraticRootsNSWC, QuadraticRootsSridharachary
  use FatesUtilsMod,     only : GetNeighborDistance

  implicit none

  ! CONSTANTS:
  integer, parameter          :: n = 4                    ! number of points to
  character(len=*), parameter :: out_file = 'quad_out.nc' ! output file

  ! LOCALS:
  integer  :: i                ! looping index
  real(r8) :: a(n), b(n), c(n) ! coefficients for quadratic
  real(r8) :: root1(n)         ! real part of first root of
  real(r8) :: root2(n)         ! real part of second root of
```

```
a = (/1.0_r8, 1.0_r8, 5.0_r8, 1.5_r8/)
b = (/-2.0_r8, 7.0_r8, 10.0_r8, 3.2_r8/)
c = (/1.0_r8, 12.0_r8, 3.0_r8, 1.1_r8/)      You, 10 months ago • add

do i = 1, n
  call QuadraticRootsNSWC(a(i), b(i), c(i), root1(i), root2(i))
end do

call WriteQuadData(out_file, n, a, b, c, root1, root2)
```

“Off-offline” testing

- Test small, isolated pieces of the model
- Validate internal model behavior before running full-scale runs

```python
def build_tests(build_dir:str, cmake_directory:str, make_j:int, clean:bool=False,
                verbose:bool=False):
    """Builds the test executables

    Args:
        build_dir (str): build directory
        cmake_directory (str): directory where the make CMakeLists.txt file is
        make_j (int): number of processes to use for make
        clean (bool, optional): whether or not to clean the build first. Defaults to False.
        verbose (bool, optional): whether or not to run make with verbose output. Defaults to False.
    """
    # create the build directory
    full_build_path = prep_build_dir(build_dir, clean=clean)

    # get cmake args and the pfunit and netcdf paths
    cmake_args = get_extra_cmake_args(full_build_path, _MPI_LIBRARY)
    pfunit_path = find_library(full_build_path, cmake_args, "PFUNIT_PATH")
```

“Off-offline” testing

- Test small, isolated pieces of the model
- Validate internal model behavior before running full-scale runs

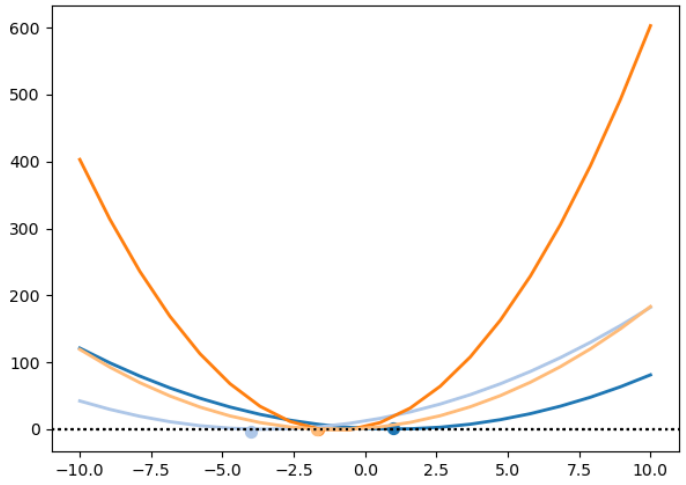
```
def build_tests(build_dir:str, cmake_directory:str, make_j:int, clean:bool=False,
                verbose:bool=False):
    """B
    Args
    """
    # cr
    full
    # ge
    cmak
    pfun
```

```
# run executables for each test in test list
if run_executables:
    print("Running executables")
    for _, test in test_dict.items():
        # prepend parameter file (if required) to argument list
        args = test.other_args
        if test.use_param_file:
            args.insert(0, param_file)
        # run
        run_fortran_exectuables(
            build_dir_path, test.test_dir, test.test_exe, run_dir_path, args
        )
```

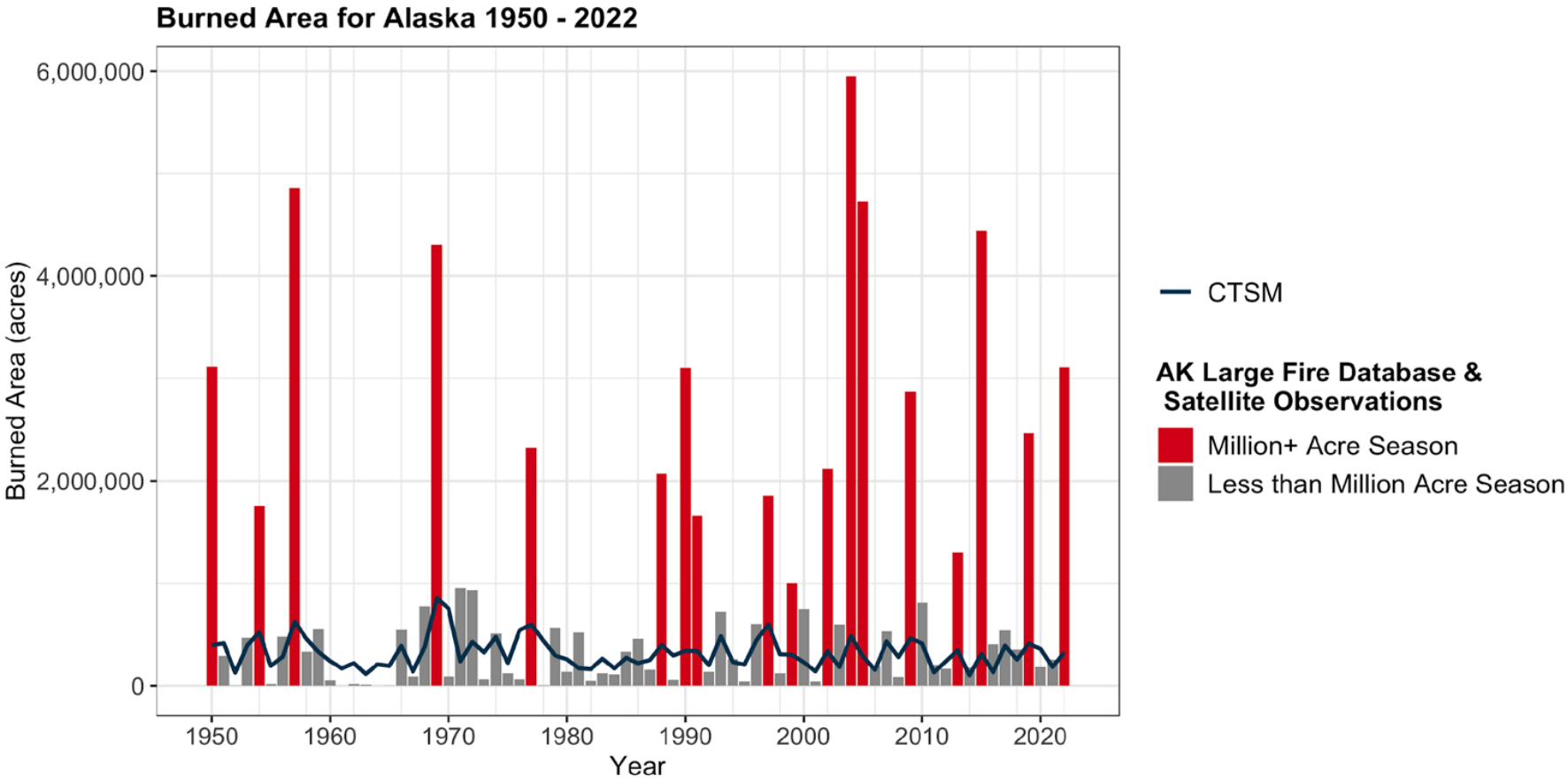
“Off-offline” testing

- Test small, isolated pieces of the model
- Validate internal model behavior before running full-scale runs

```
def plot_quad_and_roots(a_coeff, b_coeff, c_coeff, root1, root2):
    """Plots a set of quadratic formulas (ax**2 + bx + c) and their two roots

    Args:
        a_coeff (float array): set of a coefficients
        b_coeff (float array): set of b coefficients
        c_coeff (float array): set of b coefficients
        root1 (float array): set of first real roots
        root2 (float array): set of second real roots
    """
    num_equations = len(a_coeff)

    plt.figure(figsize=(7, 5))
    x_vals = np.linspace(-10.0, 10.0, num=20)

    colors = get_color_palette(num_equations)
    for i in range(num_equations):
        y_vals = a_coeff[i]*x_vals**2 + b_coeff[i]*x_vals + c_coeff[i]
        plt.plot(x_vals, y_vals, lw=2, color=colors[i])
        plt.scatter(root1[i], root2[i], color=colors[i], s=50)
        plt.axhline(y=0.0, color='k', linestyle='dotted')
```

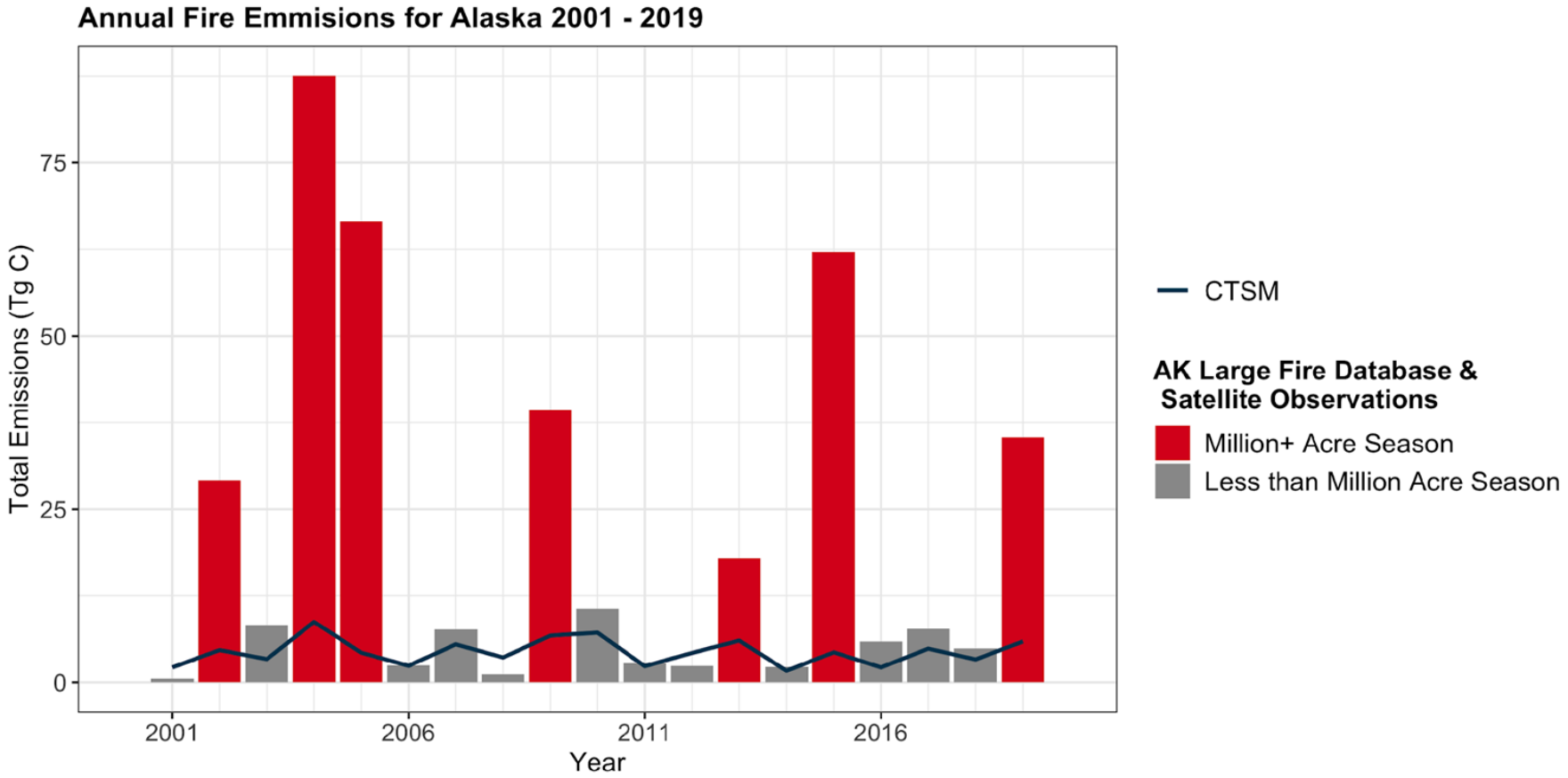


Example – burned area and fire emissions



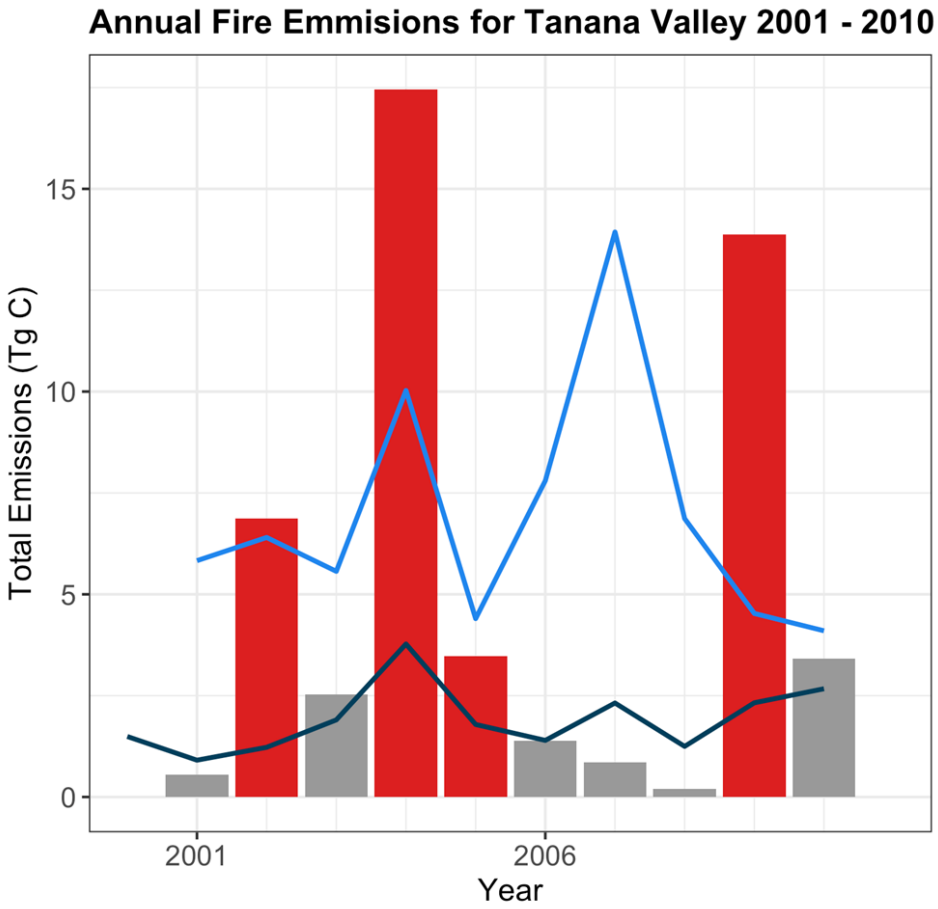
Potter et al. 2022, *ORNL DAAC*
Lindgren et al. 2015, *SNAP*

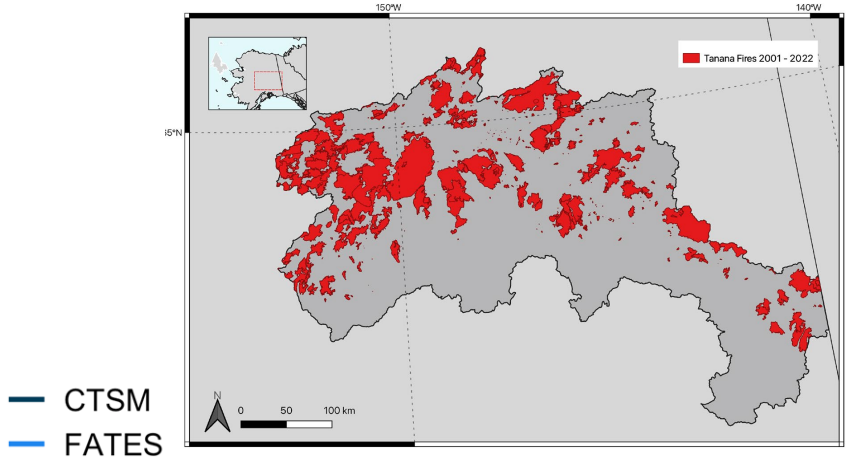
Example – burned area and fire emissions



Potter et al. 2022, *ORNL DAAC*

Examples – burned area and fire emissions



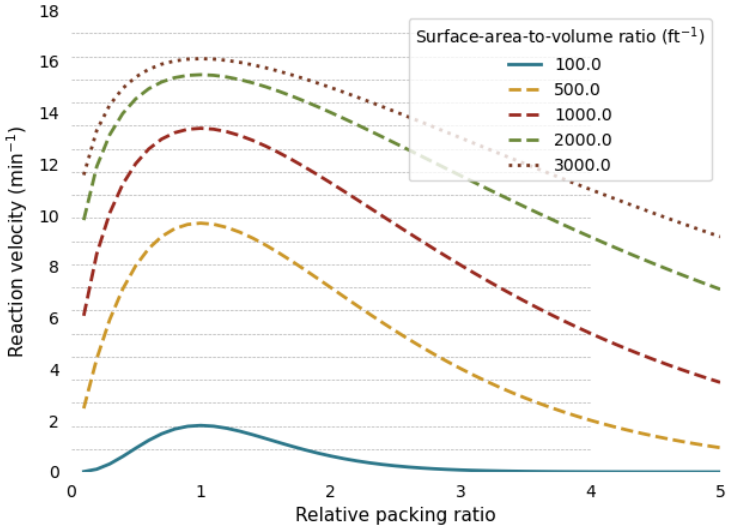


AK Large Fire Database & Satellite Observations

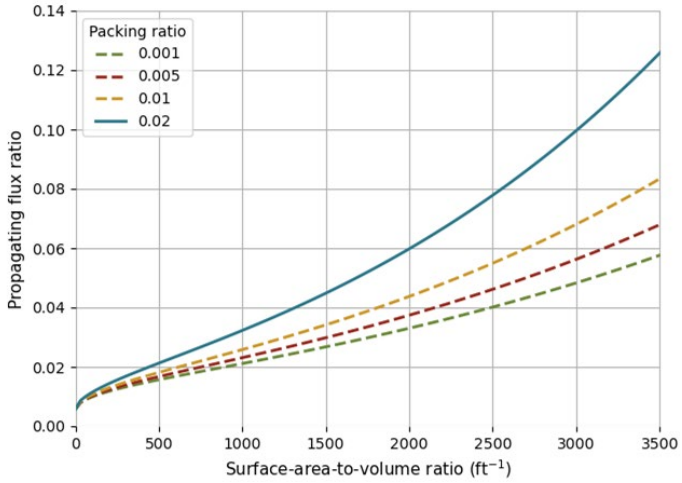
- Million+ Acre Season
- Less than Million Acre Season

Potter et al. 2022, *ORNL DAAC*

Testing a few fire equations

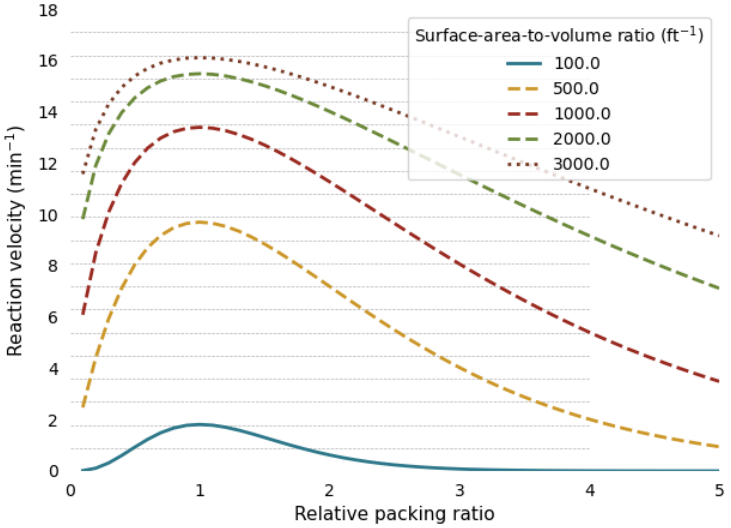



```
do i = 1, num_beta_r

  beta_ratio(i) = beta_r_min + beta_r_inc*(i-1)

  do j = 1, size(SAV_vals)
    SAV(j) = SAV_vals(j)/30.48_r8 ! convert from /ft to /cm
    max_reaction_vel =  MaximumReactionVelocity(SAV(j))
    reaction_velocity(i,j) = OptimumReactionVelocity(max_reaction_vel, SAV(j),
      beta_ratio(i))
  end do
end do
```

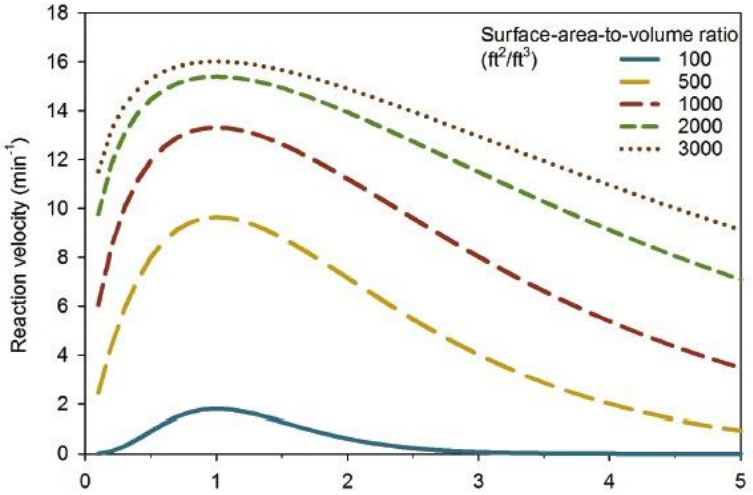


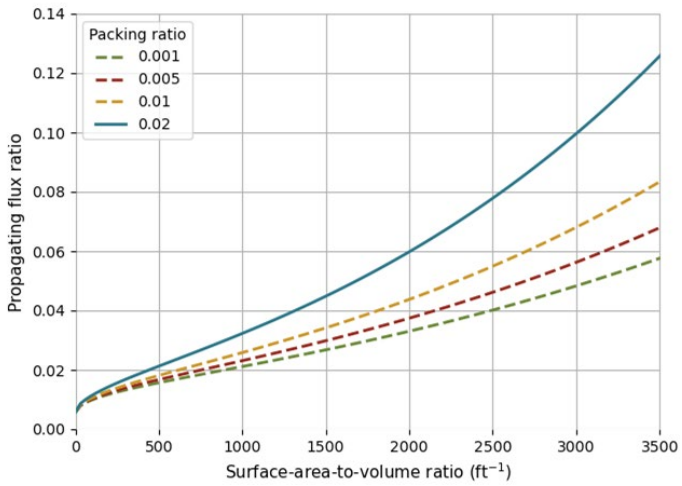

```
do i = 1, num_SAV

  SAV(i) = SAV_min + SAV_inc*(i-1)

  do j = 1, size(packing_ratio)
    beta(j) = packing_ratio(j)
    propagating_flux(i,j) = PropagatingFlux(packing_ratio(j), SAV(i))
  end do
end do
```

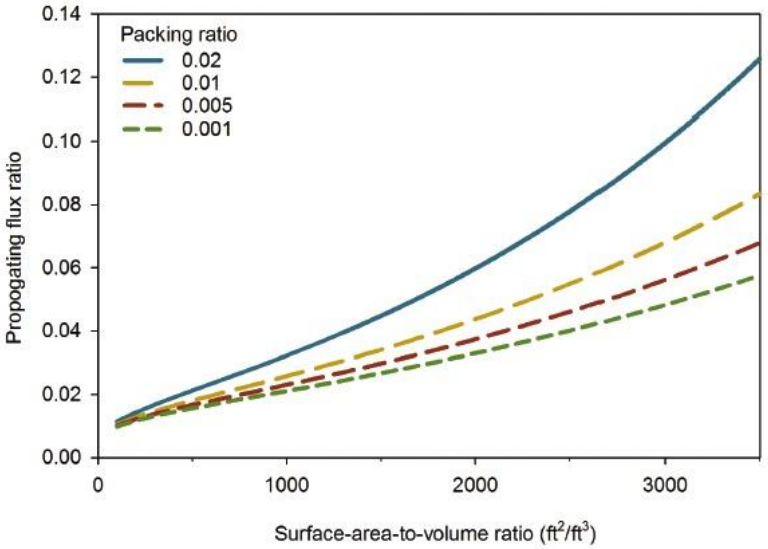
Testing a few fire equations



Andrews et al. 2018







Synthetic fuel types

Standard Fire Behavior Fuel Models: A Comprehensive Set for Use with Rothermel's Surface Fire Spread Model

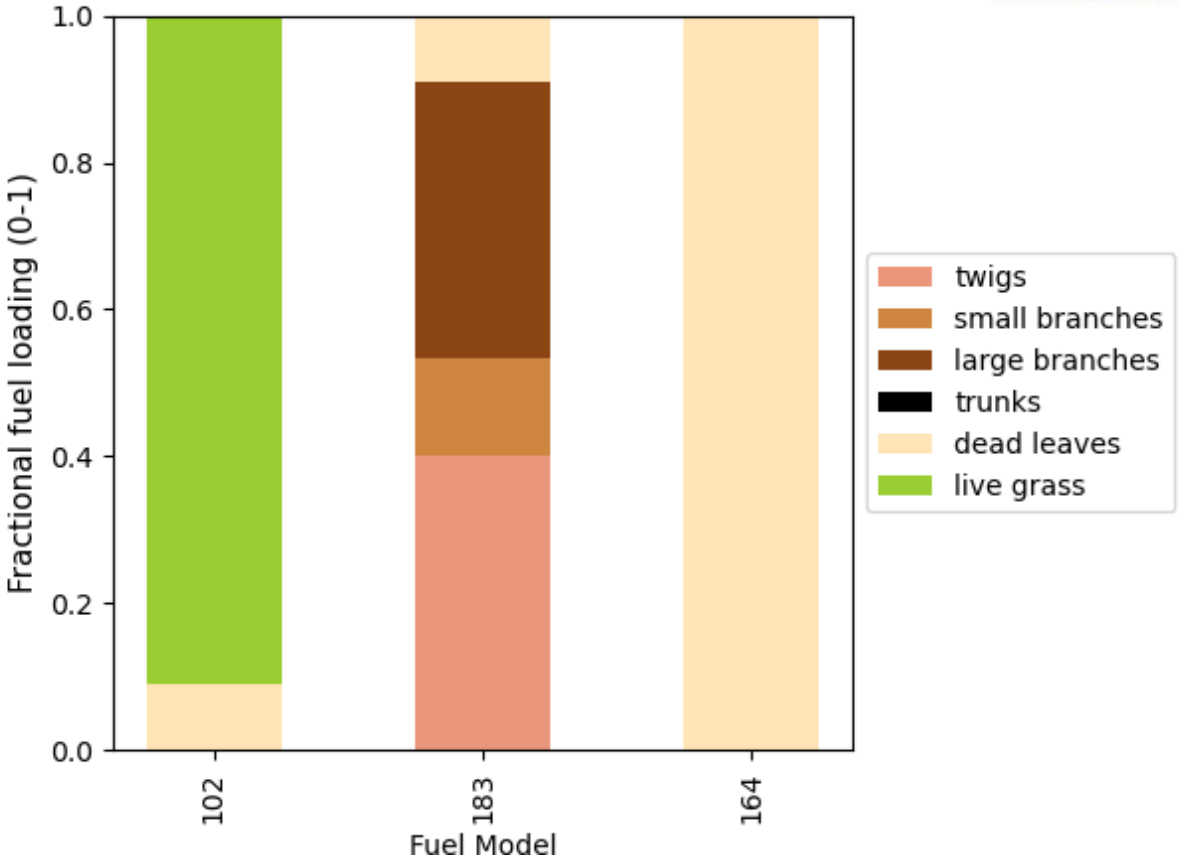
Joe H. Scott
Robert E. Burgan

183: "moderate load conifer litter"

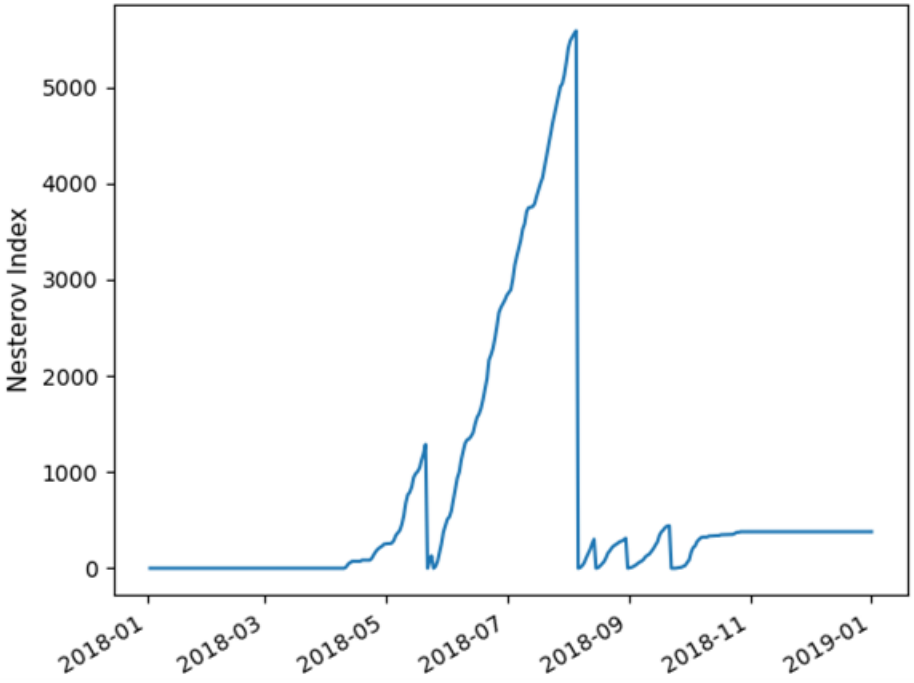
164: "dwarf conifer with understory"

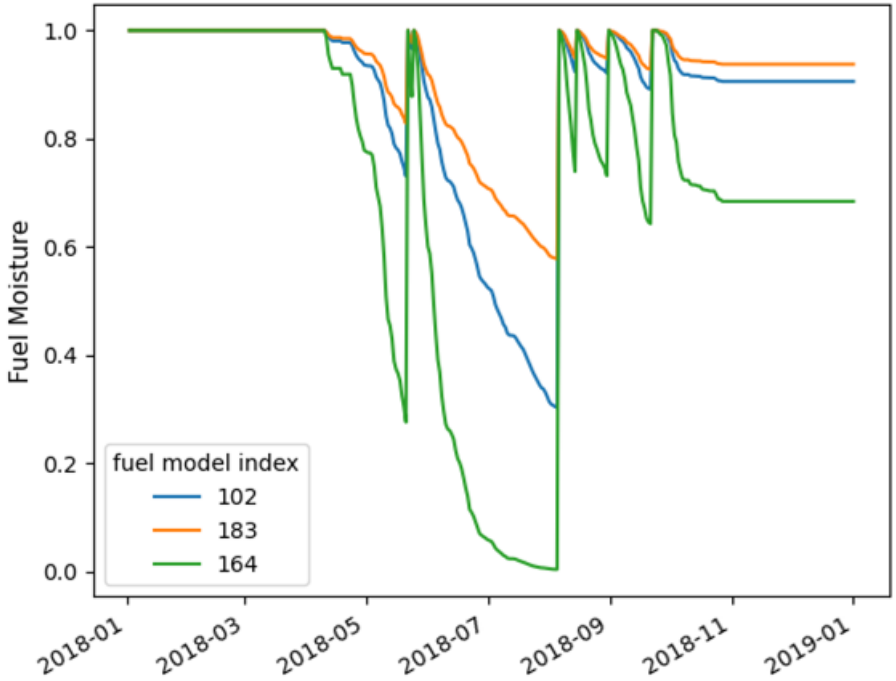
102: "low load dry climate grass"

Synthetic fuel types

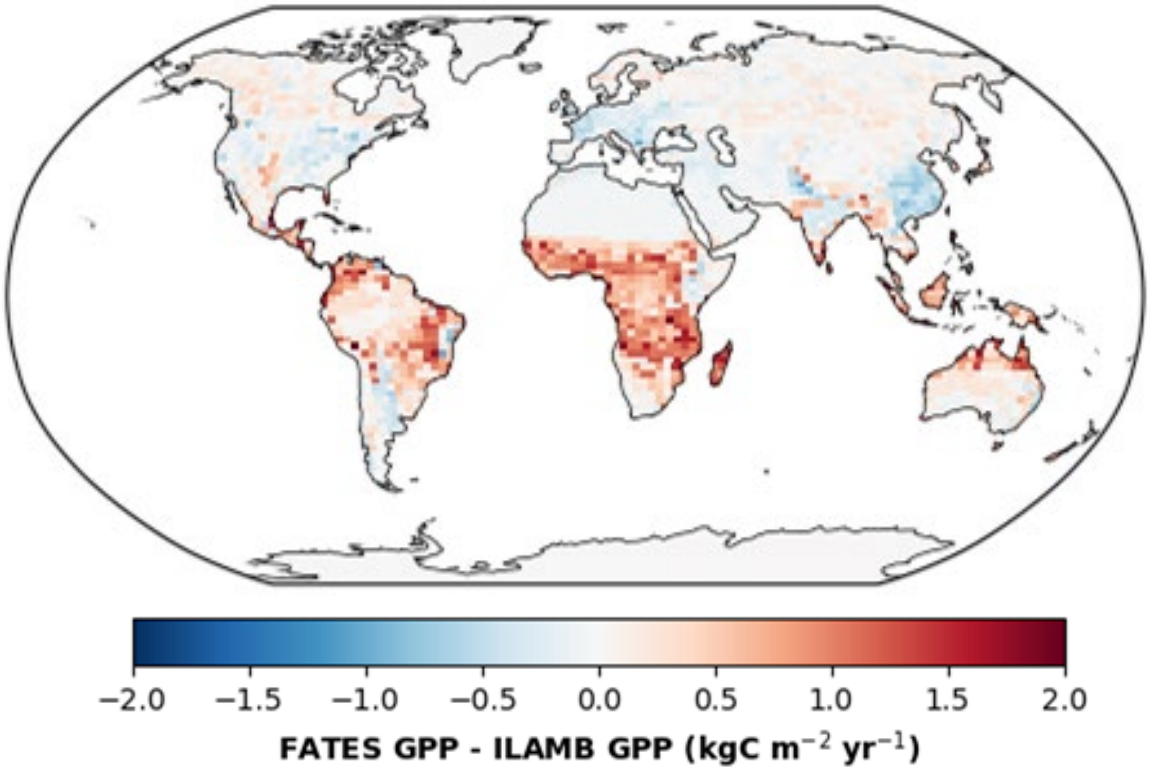


Testing fuel moisture





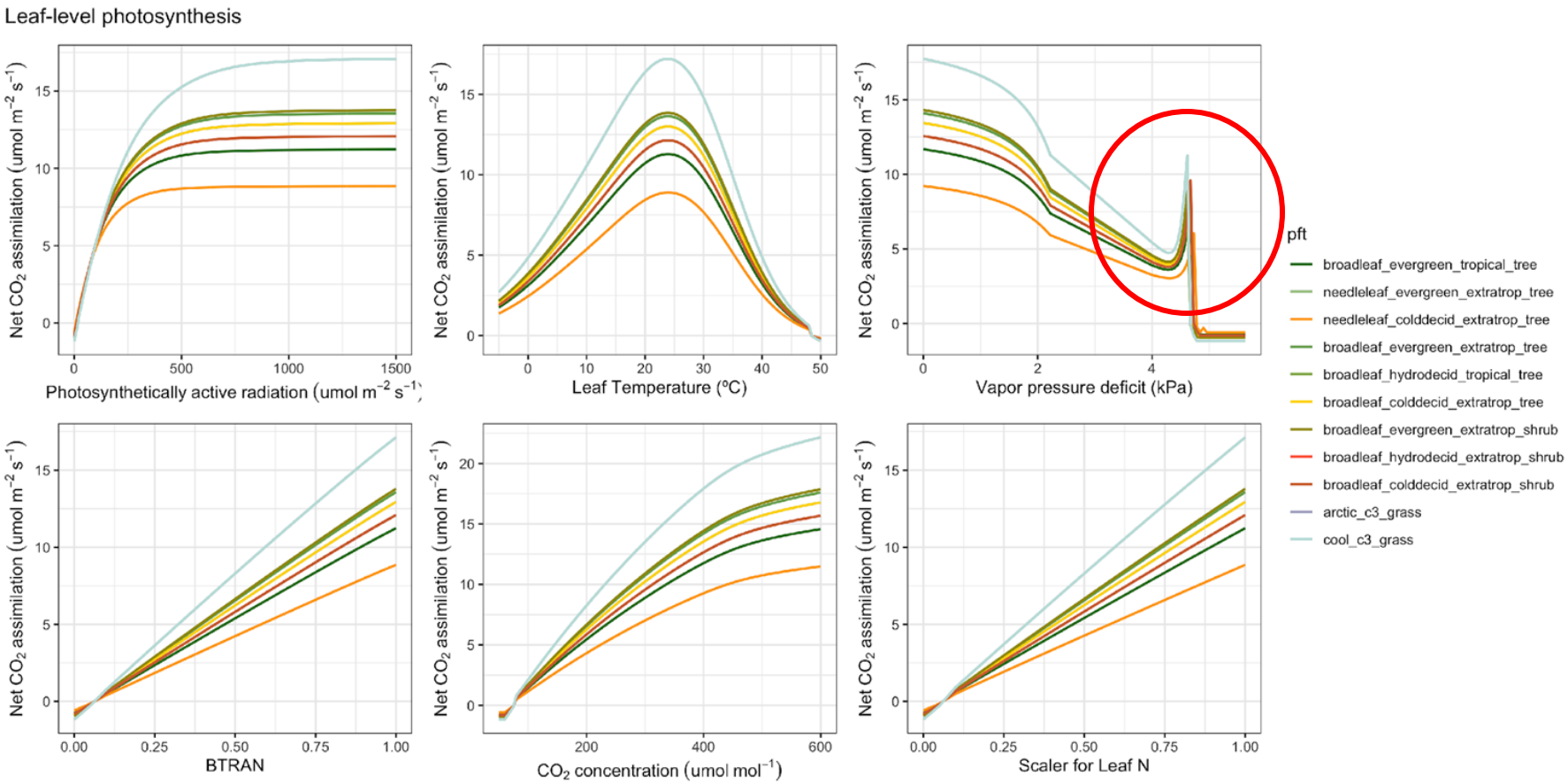
Situation – FATES GPP is too high!



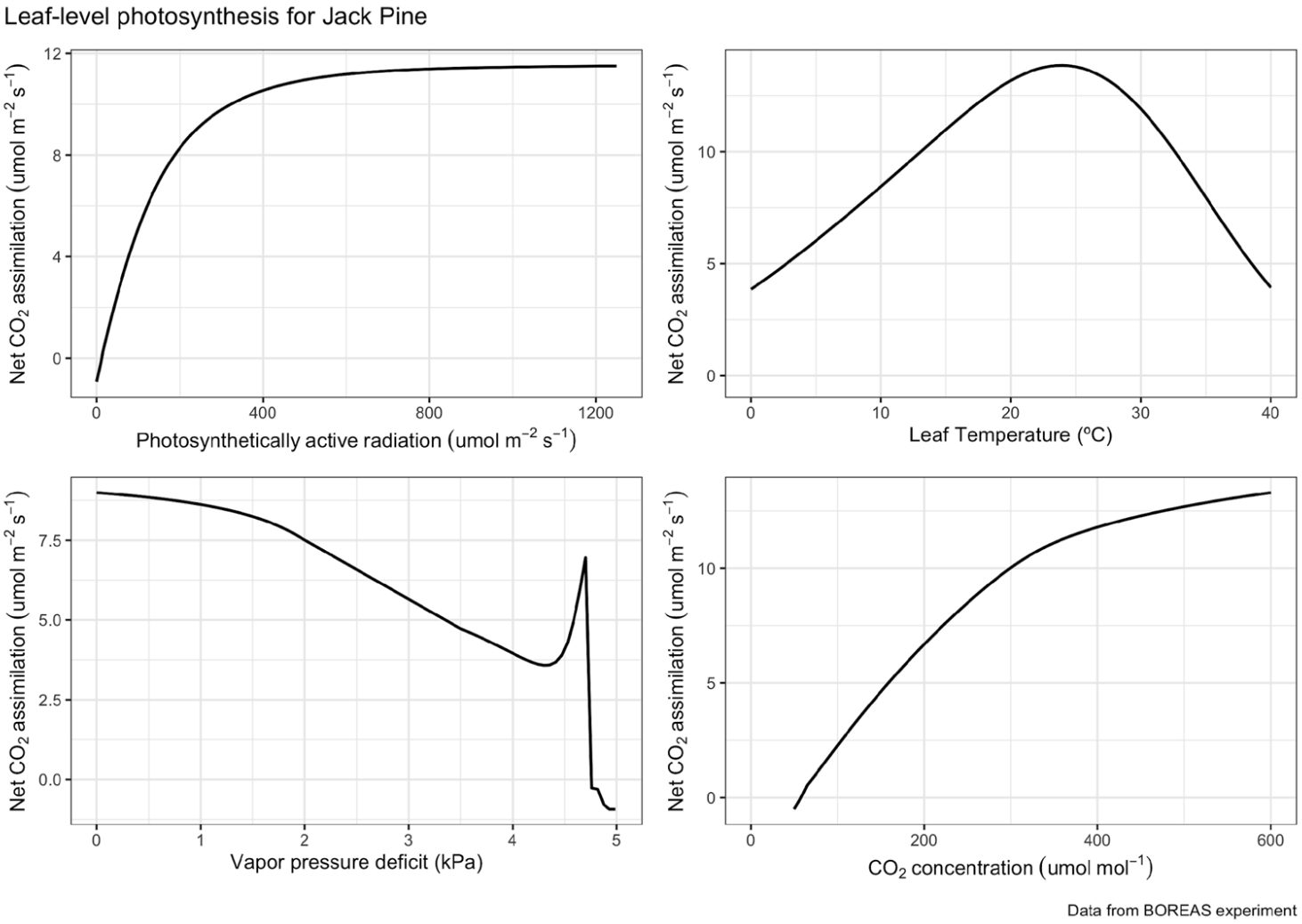
Testing leaf -level photosynthesis

```
! calculate canopy gas parameters
call GetCanopyGasParameters(can_air_press, can_o2_pp, veg_tempk, can_tempk,          &
  can_vpress, veg_esat, leaf_bl_resistance, mm_kco2, mm_ko2, co2_compensation_pt,    &
  cf, leaf_bl_conductance, can_vpress_constrained)

! calculate leaf biophysical rates
call LeafLayerBiophysicalRates(par, ft, EDPftvarcon_inst%vcmax25top(ft,1),           &
  param_derived%jmax25top(ft,1), param_derived%kp25top(ft,1), nscaler, veg_tempk,     &
  dayl_fact, can_tempk, can_tempk, btran, vcmax, jmax, co2_rcurve_islope)

! pulled out from model for now
stomatal_int_btran = max(cf/2.E8_r8, EDPftvarcon_inst%stomatal_intercept(ft)*btran)
      You, 7 months ago • start adding other scalers
! calculate leaf-level photosynthesis
call LeafLayerPhotosynthesis(1.0_r8, par, 0.0_r8, 10.0_r8, 0.0_r8, ft, vcmax, jmax,  &
  co2_rcurve_islope, veg_tempk, veg_esat, can_air_press, can_co2_pp, can_o2_pp, btran, &
  stomatal_int_btran, cf, leaf_bl_conductance, can_vpress_constrained, mm_kco2,      &
  mm_ko2, co2_compensation_pt, leaf_maintenance_resp, 999.0_r8, leaf_bl_resistance,  &
  assimilation, stomatal_resistance, leaf_photo, c13disc, test_out)
```

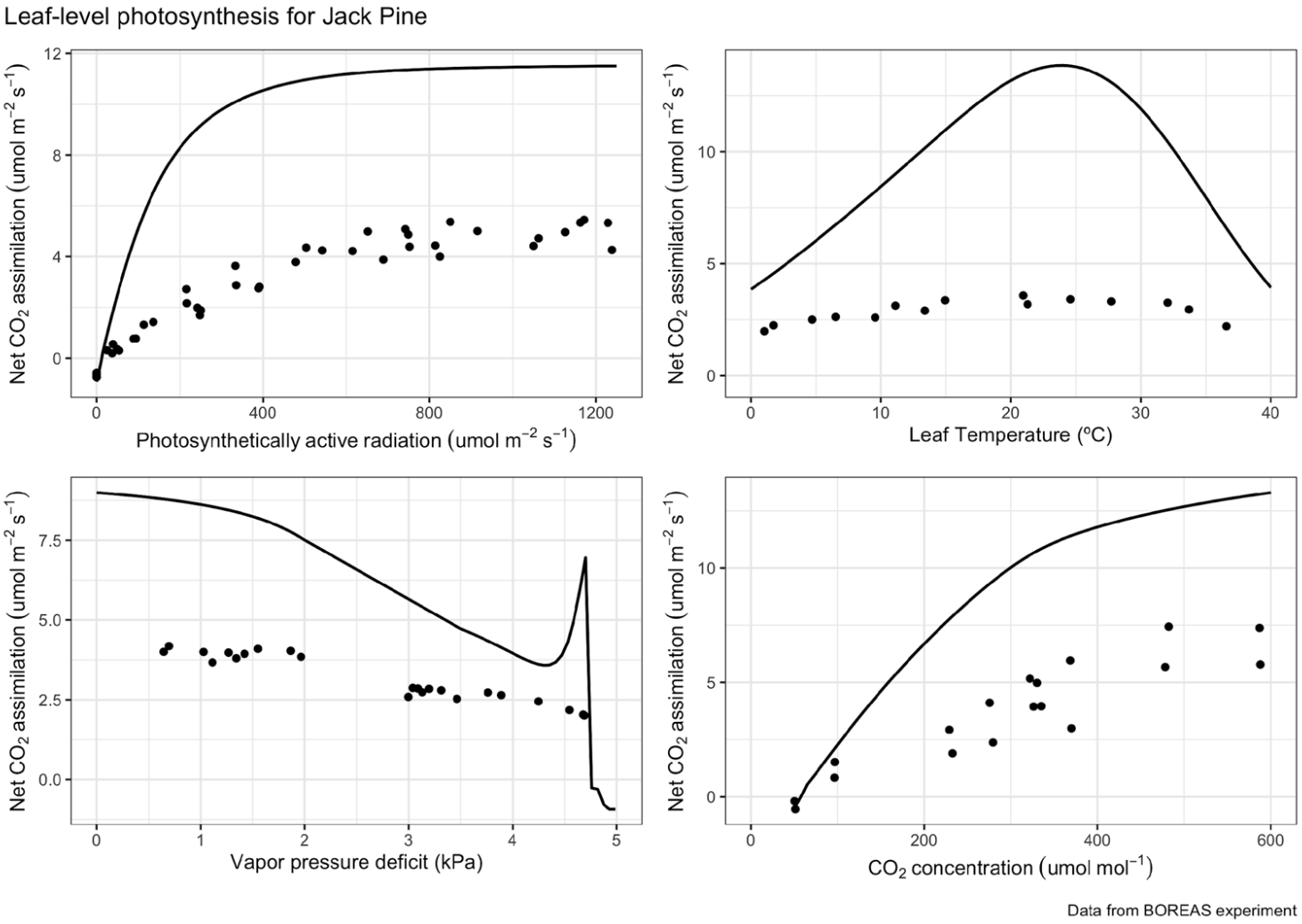
Testing leaf -level photosynthesis



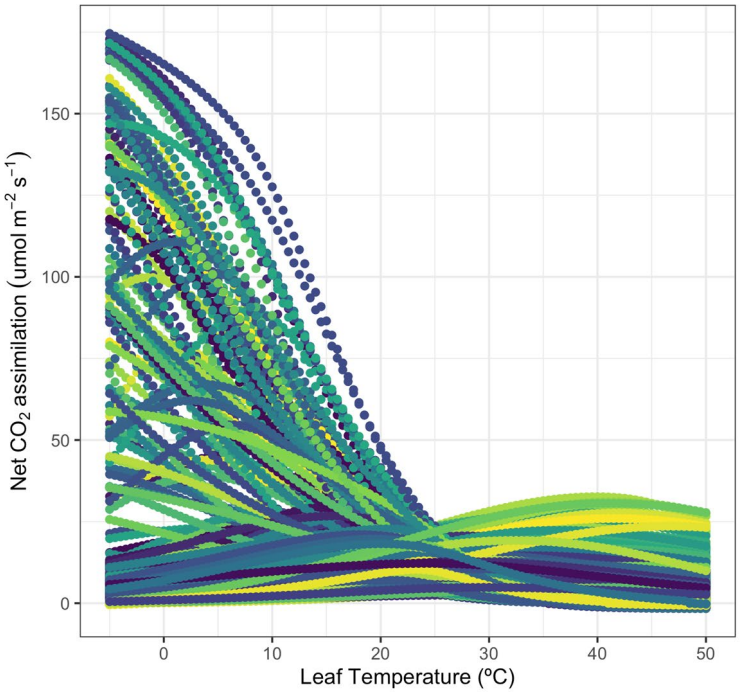
Testing leaf -level photosynthesis

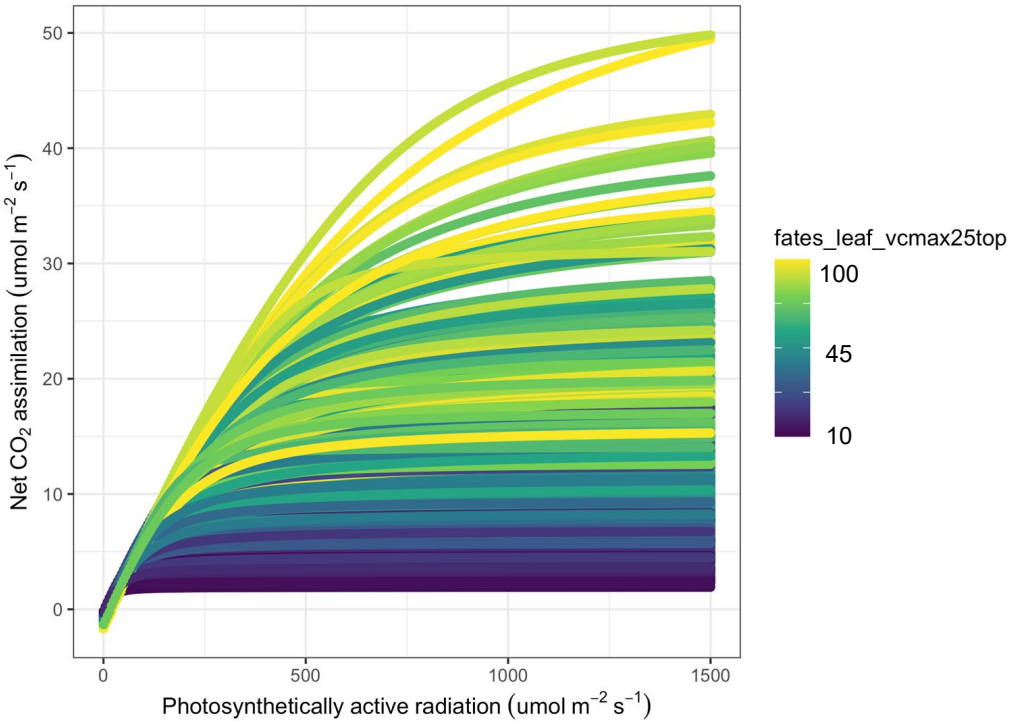


Testing leaf -level photosynthesis

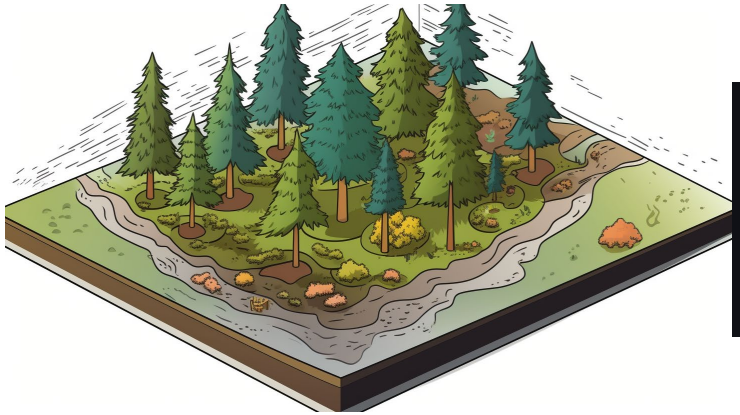


Testing leaf -level photosynthesis





Synthetic patches



even-aged evergreen

```
call this%AddPatch(patch_id=2, patch_name='evergreen', area=500.0_r8,
  ages=(/50.0_r8, 50.0_r8/),
  dbhs=(/30.0_r8, 25.0_r8/),
  densities=(/0.015_r8, 0.015_r8/),
  pft_ids=(/2, 2/),
  canopy_layers=(/1, 1/))
```

tropical

```
call this%AddPatch(patch_id=1, patch_name='tropical', area=500.0_r8,
  ages=(/100.0_r8, 80.0_r8, 40.0_r8, 20.0_r8/),
  dbhs=(/60.0_r8, 50.0_r8, 25.0_r8, 10.0_r8/),
  densities=(/0.005_r8, 0.008_r8, 0.02_r8, 0.017_r8/),
  pft_ids=(/1, 1, 1, 1/),
  canopy_layers=(/1, 1, 2, 2/))
```

Difference from Prototyping

```
do i = 1, num_SA
  SAV(i) = SAV_min + SAV_in
  do j =
    beta(
    propag
  end do
end do
```

```
size(packing_rat
= packing_ratio
ng_flux(i,j)
```

Functional Testing

- Verifies existing model behavior
- Isolates components to check expected results
- Helps detect unexpected changes before full runs
- Uses ***production code***

Prototyping

- Tests new ideas or formulations
- Explores feasibility before integration
- ***Usually in higher-level language***

Want to try FATES functional testing?

FATES repo: https://github.com/NGEET/fates

- see README.testing.md in **testing** directory

Requirements:

- python environment
- cime & shr repositories (can get via git-fleximod)
- works OOB on derecho
- see cime_setup.md for personal computer use

First Steps:

- Run a test: `./run_functional_tests.py`
- Modify a test case or use a new parameter file and observe output

```
(base)
~/Documents/ncar/CTSM/src/fates/testing ± main
❯ ./run_functional_tests.py -t allometry --save-figs
```