1 Outputs produced by the CLASSIC benchmark framework

Table S1. Statistics of CLASSIC simulated RNS against FLUXNET observations. RMSE (in W m^{-2})and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	37.463	0.706
BR-Sa1	EBF	19.139	0.146
FR-Pue	EBF	11.356	0.968
GF-Guy	EBF	13.576	0.594
GH-Ank	EBF	24.739	-0.127
MY-PSO	EBF	15.12	-0.211
CA-Qfo	ENF	8.39	0.975
CZ-BK1	ENF	17.917	0.885
DE-Tha	ENF	10.329	0.968
FI-Hyy	ENF	8.895	0.974
IT-Lav	ENF	12.625	0.966
IT-SRo	ENF	32.33	0.826
NL-Loo	ENF	6.098	0.99
RU-Fyo	ENF	15.836	0.931
CA-TPD	DBF	11.77	0.962
DK-Sor	DBF	7.238	0.984
FR-Fon	DBF	19.794	0.881
US-WCr	DBF	14.094	0.921
ZM-Mon	DBF	35.981	-1.582
CG-Tch	SAV	20.521	0.094
SD-Dem	SAV	15.202	0.552
ZA-Kru	SAV	32.45	0.306
CN-Dan	GRA	17.595	0.871
IT-Tor	GRA	25.827	0.857
PA-SPs	GRA	15.344	0.444
RU-Ha1	GRA	21.409	0.834
US-Wkg	GRA	12.407	0.911
DE-Kli	CRO	9.195	0.971
ES-LgS	OSH	41.291	0.588
RU-Che	WET	26.225	0.749
RU-SkP	DNF	34.616	0.67

Table S2. Statistics of CLASSIC simulated HFLS against FLUXNET observations. RMSE (in W m^{-2}) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	38.84	-0.155
FR-Pue	EBF	17.264	0.553
GH-Ank	EBF	33.97	-0.27
MY-PSO	EBF	33.972	-21.619
CA-Qfo	ENF	21.19	0.077
DE-Tha	ENF	29.209	-0.163
FI-Hyy	ENF	24.056	0.193
IT-Lav	ENF	21.906	0.127
IT-SRo	ENF	42.414	-0.443
NL-Loo	ENF	36.341	-0.801
RU-Fyo	ENF	22.074	0.549
CA-TPD	DBF	30.234	0.454
DK-Sor	DBF	34.76	0.299
US-WCr	DBF	23.801	0.55
ZM-Mon	DBF	47.782	-0.605
DE-Kli	CRO	27.613	0.317
ES-LgS	OSH	16.385	0.046
IT-Tor	GRA	73.755	-0.076
RU-Ha1	GRA	37.294	-0.772
US-Wkg	GRA	17.586	0.423
RU-SkP	DNF	24.642	0.314
SD-Dem	SAV	43.059	-0.868
ZA-Kru	SAV	40.427	-0.24

Table S3. Statistics of CLASSIC simulated HFSS against FLUXNET observations. RMSE (in W m^{-2}) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	17.851	0.511
FR-Pue	EBF	18.42	0.855
GH-Ank	EBF	24.128	-1.163
MY-PSO	EBF	16.648	-1.69
CA-Qfo	ENF	15.236	0.826
DE-Tha	ENF	19.401	0.61
FI-Hyy	ENF	19.359	0.634
IT-Lav	ENF	16.458	0.84
IT-SRo	ENF	32.831	0.749
NL-Loo	ENF	37.758	0.252
RU-Fyo	ENF	15.457	0.798
CA-TPD	DBF	28.451	0.359
DK-Sor	DBF	35.824	-0.468
US-WCr	DBF	35.281	-1.103
ZM-Mon	DBF	27.048	-0.585
DE-Kli	CRO	21.056	0.355
ES-LgS	OSH	56.334	-0.225
IT-Tor	GRA	34.887	0.158
RU-Ha1	GRA	21.299	0.094
US-Wkg	GRA	11.138	0.849
RU-SkP	DNF	26.826	0.508
SD-Dem	SAV	21.345	-1.731
ZA-Kru	SAV	20.764	-0.247

Table S4. Statistics of CLASSIC simulated HFG against FLUXNET observations. RMSE (in W m^{-2}) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	4.43	-0.83
FR-Pue	EBF	5.657	-0.984
GH-Ank	EBF	1.107	-2.434
MY-PSO	EBF	1.881	-16.98
CA-Qfo	ENF	5.768	-2.841
DE-Tha	ENF	3.222	0.156
FI-Hyy	ENF	3.193	0.634
IT-Lav	ENF	5.399	-1.129
IT-SRo	ENF	3.946	-0.624
NL-Loo	ENF	2.532	-0.197
RU-Fyo	ENF	5.693	-2.425
CA-TPD	DBF	4.144	-0.097
DK-Sor	DBF	3.004	0.331
US-WCr	DBF	9.097	-1.209
ZM-Mon	DBF	2.879	-0.915
DE-Kli	CRO	6.463	0.24
ES-LgS	OSH	8.141	-0.599
IT-Tor	GRA	5.592	-0.801
RU-Ha1	GRA	14.062	-20.028
US-Wkg	GRA	2.404	0.702
RU-Che	WET	16.869	-31.633
RU-SkP	DNF	12.185	-37.282
SD-Dem	SAV	2.192	0.081
ZA-Kru	SAV	7.85	-0.043

Table S5. Statistics of CLASSIC simulated GPP against FLUXNET observations. RMSE (in kg C m^{-2} $month^{-1}$) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	0.14	-1.371
BR-Sa1	EBF	0.082	-0.291
FR-Pue	EBF	0.053	-0.235
GF-Guy	EBF	0.078	-3.641
GH-Ank	EBF	0.063	-0.141
MY-PSO	EBF	0.055	-5.062
CA-Qfo	ENF	0.076	-0.244
CZ-BK1	ENF	0.042	0.847
DE-Tha	ENF	0.037	0.902
FI-Hyy	ENF	0.055	0.704
IT-Lav	ENF	0.07	0.652
IT-SRo	ENF	0.064	0.347
NL-Loo	ENF	0.045	0.753
RU-Fyo	ENF	0.044	0.859
CA-TPD	DBF	0.053	0.838
DK-Sor	DBF	0.055	0.884
FR-Fon	DBF	0.06	0.805
US-WCr	DBF	0.07	0.739
ZM-Mon	DBF	0.087	-0.352
CG-Tch	SAV	0.192	-3.422
SD-Dem	SAV	0.09	0.026
ZA-Kru	SAV	0.078	0.076
CN-Dan	GRA	0.025	0.3
IT-Tor	GRA	0.069	0.497
PA-SPs	GRA	0.084	0.009
RU-Ha1	GRA	0.058	-0.039
US-Wkg	GRA	0.035	-0.337
DE-Kli	CRO	0.142	-0.28
ES-LgS	OSH	0.019	0.581
RU-Che	WET	0.047	0.115
RU-SkP	DNF	0.042	0.118

Table S6. Statistics of CLASSIC simulated RECO against FLUXNET observations. RMSE (in kg C m^{-2} $month^{-1}$) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	0.107	-0.707
BR-Sa1	EBF	0.103	-0.946
FR-Pue	EBF	0.038	0.136
GF-Guy	EBF	0.075	-0.565
GH-Ank	EBF	0.103	-1.459
MY-PSO	EBF	0.049	-1.335
CA-Qfo	ENF	0.062	-0.343
CZ-BK1	ENF	0.086	-1.606
DE-Tha	ENF	0.058	0.341
FI-Hyy	ENF	0.066	-0.059
IT-Lav	ENF	0.098	-5.146
IT-SRo	ENF	0.061	0.177
NL-Loo	ENF	0.068	-0.33
RU-Fyo	ENF	0.046	0.796
CA-TPD	DBF	0.115	-1.68
DK-Sor	DBF	0.052	0.641
FR-Fon	DBF	0.079	-1.726
US-WCr	DBF	0.062	0.311
ZM-Mon	DBF	0.096	-0.786
CG-Tch	SAV	0.157	-4.783
SD-Dem	SAV	0.039	0.22
ZA-Kru	SAV	0.072	-0.03
CN-Dan	GRA	0.012	0.645
IT-Tor	GRA	0.025	0.554
PA-SPs	GRA	0.046	0.541
RU-Ha1	GRA	0.02	0.802
US-Wkg	GRA	0.033	-0.68
DE-Kli	CRO	0.047	0.415
ES-LgS	OSH	0.014	0.137
RU-Che	WET	0.012	0.767
RU-SkP	DNF	0.077	-10.344

Table S7. Statistics of CLASSIC simulated NEE against FLUXNET observations. RMSE (in kg C m^{-2} $month^{-1}$) and R^2 are calculated as described in Section 5.1.

site	biome	RMSE	R^2
AU-Tum	EBF	0.084	-1.272
BR-Sa1	EBF	0.042	-0.674
FR-Pue	EBF	0.041	-1.268
GF-Guy	EBF	0.055	-4.653
GH-Ank	EBF	0.066	-2.909
MY-PSO	EBF	0.091	-25.714
CA-Qfo	ENF	0.031	-1.149
CZ-BK1	ENF	0.085	-0.261
DE-Tha	ENF	0.064	-0.402
FI-Hyy	ENF	0.035	0.438
IT-Lav	ENF	0.171	-3.031
IT-SRo	ENF	0.078	-2.437
NL-Loo	ENF	0.048	-0.268
RU-Fyo	ENF	0.035	-0.227
CA-TPD	DBF	0.082	-0.388
DK-Sor	DBF	0.077	0.11
FR-Fon	DBF	0.105	-0.118
US-WCr	DBF	0.073	0.132
ZM-Mon	DBF	0.047	-2.858
CG-Tch	SAV	0.084	-7.685
SD-Dem	SAV	0.067	-0.446
ZA-Kru	SAV	0.047	-0.815
CN-Dan	GRA	0.022	-1.153
IT-Tor	GRA	0.066	0.055
PA-SPs	GRA	0.083	-4.048
RU-Ha1	GRA	0.044	-3.509
US-Wkg	GRA	0.014	0.094
DE-Kli	CRO	0.114	-0.582
ES-LgS	OSH	0.023	-0.663
RU-Che	WET	0.032	-0.136
RU-SkP	DNF	0.057	-1.809

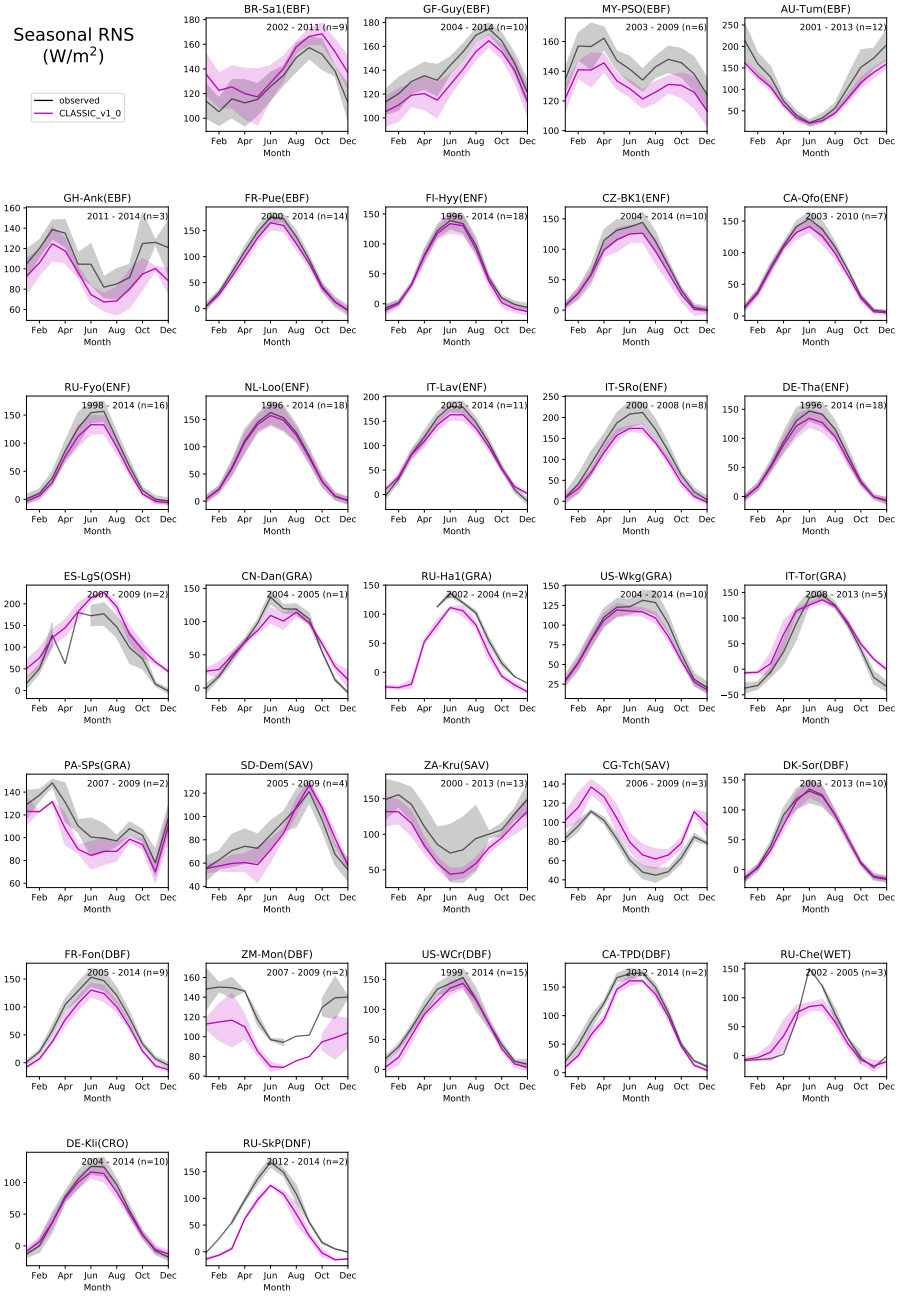

Figure S1. The mean seasonal cycle of RNS for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

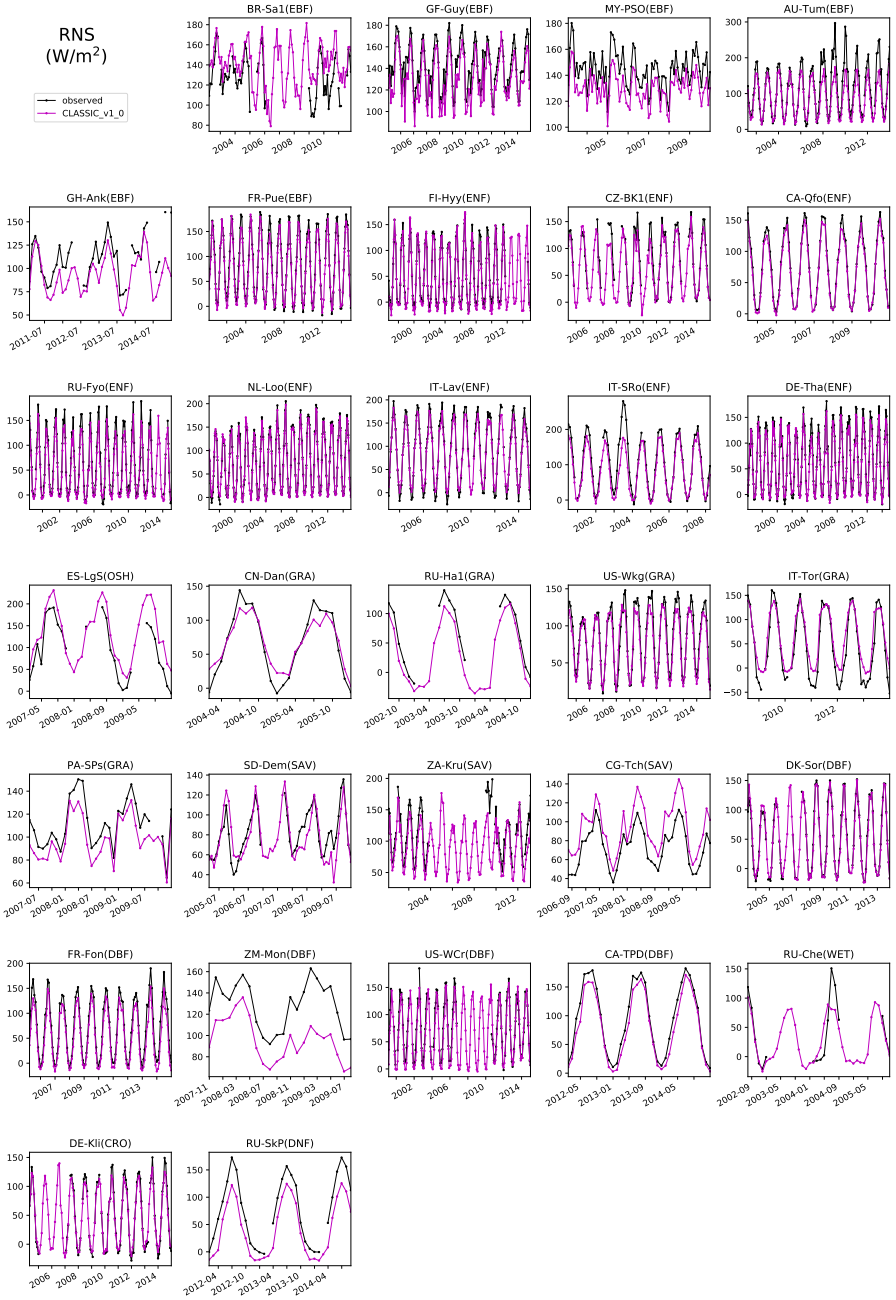

Figure S2. Timeseries of RNS for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

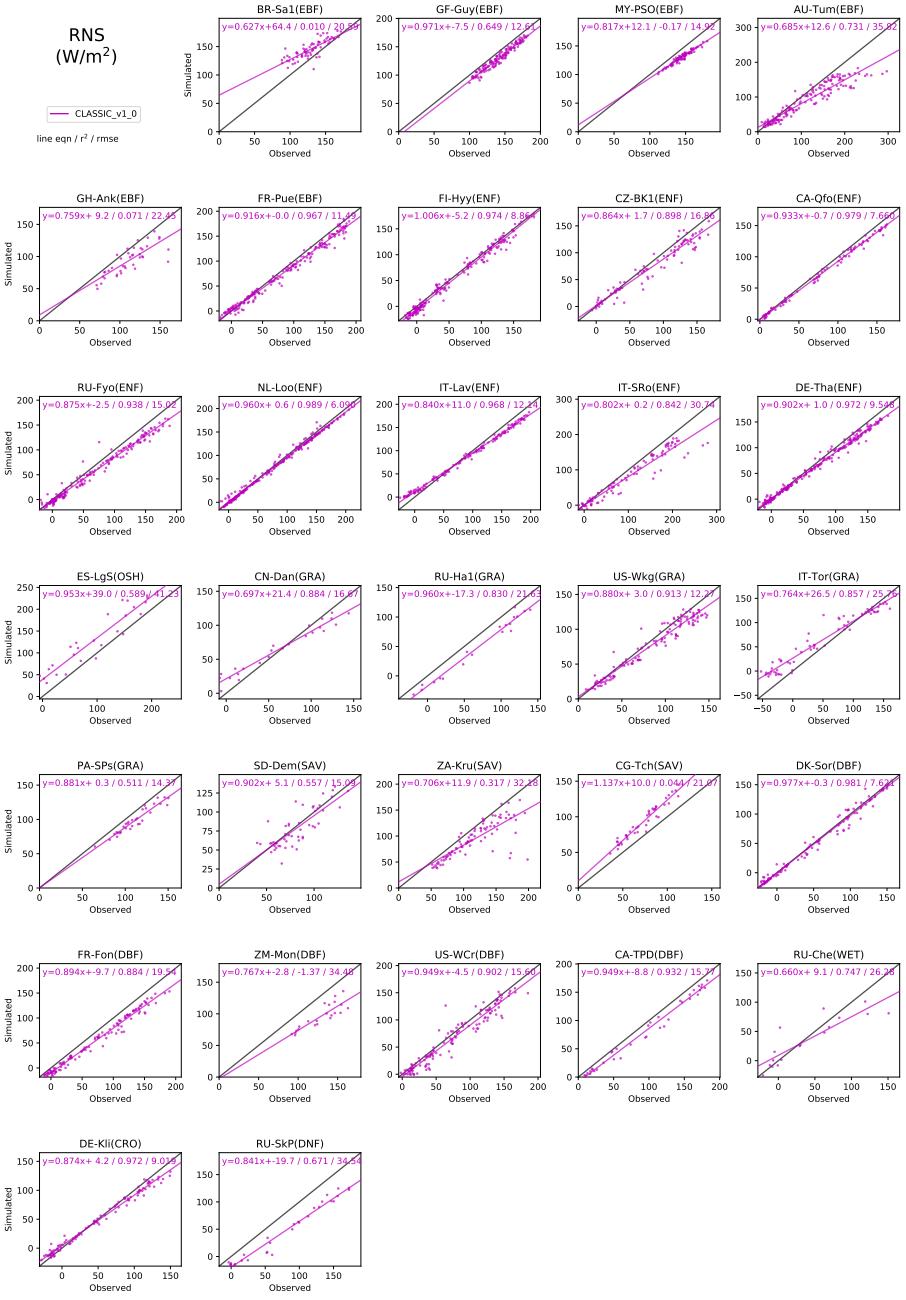

Figure S3. Scatter plot of simulated and observed RNS for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

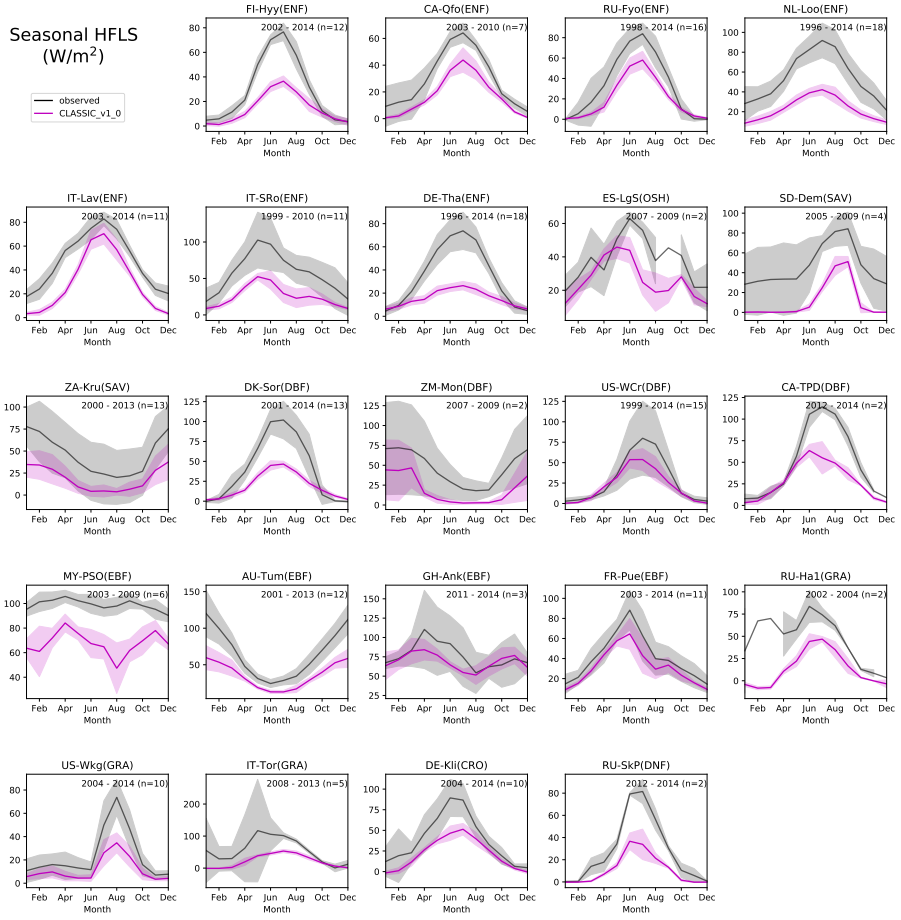

Figure S4. The mean seasonal cycle of HFLS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

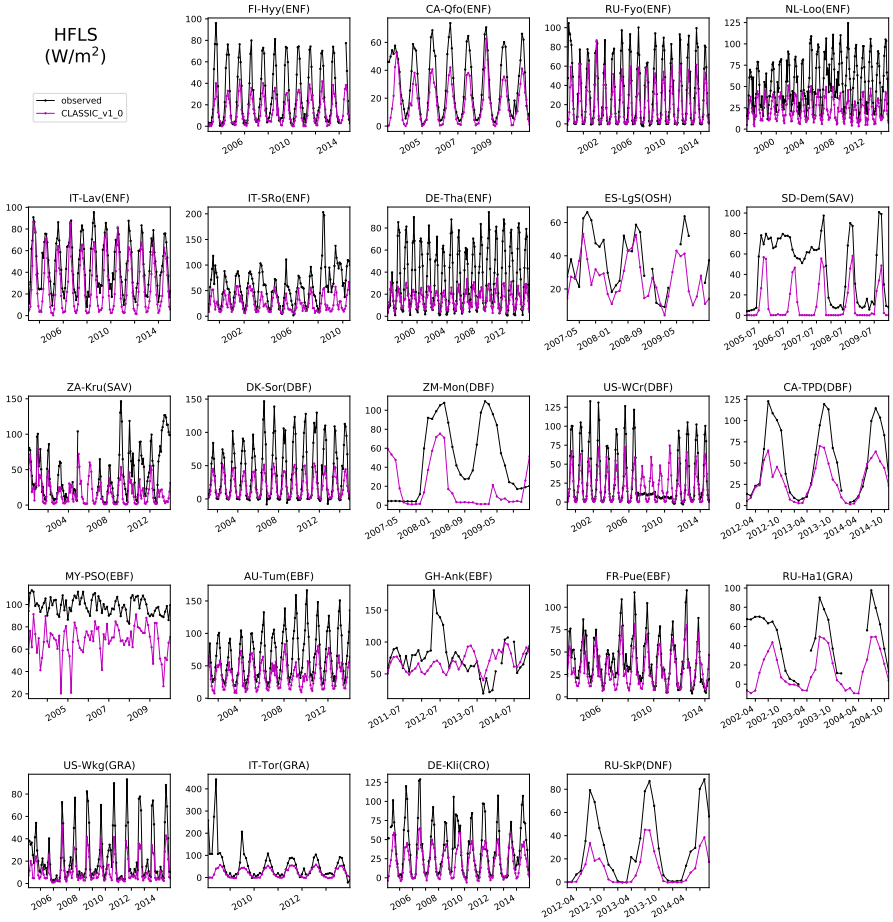

Figure S5. Timeseries of HFLS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

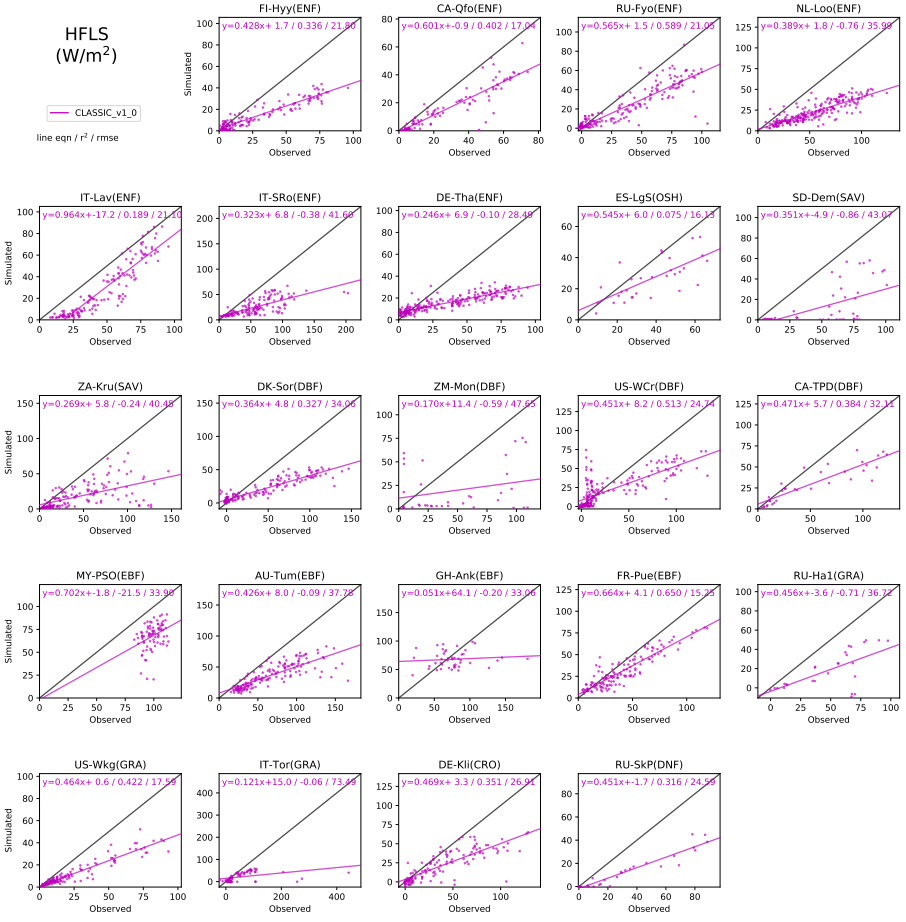

Figure S6. Scatter plot of simulated and observed HFLS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

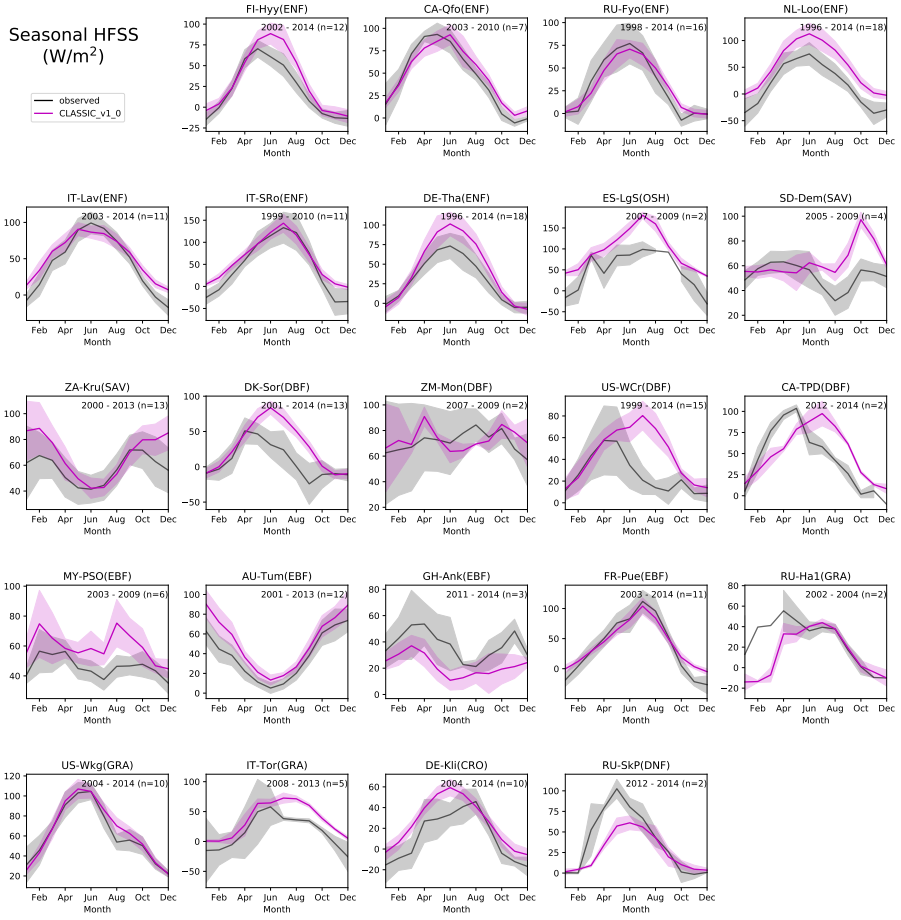

Figure S7. The mean seasonal cycle of HFSS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

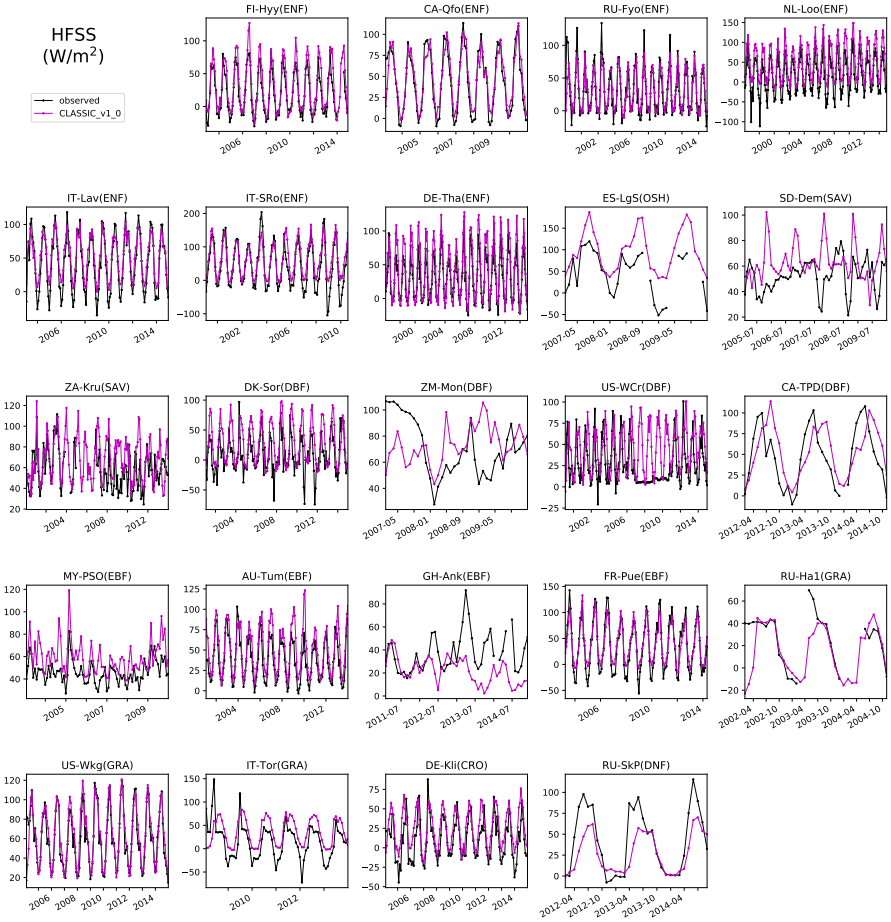

Figure S8. Timeseries of HFSS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

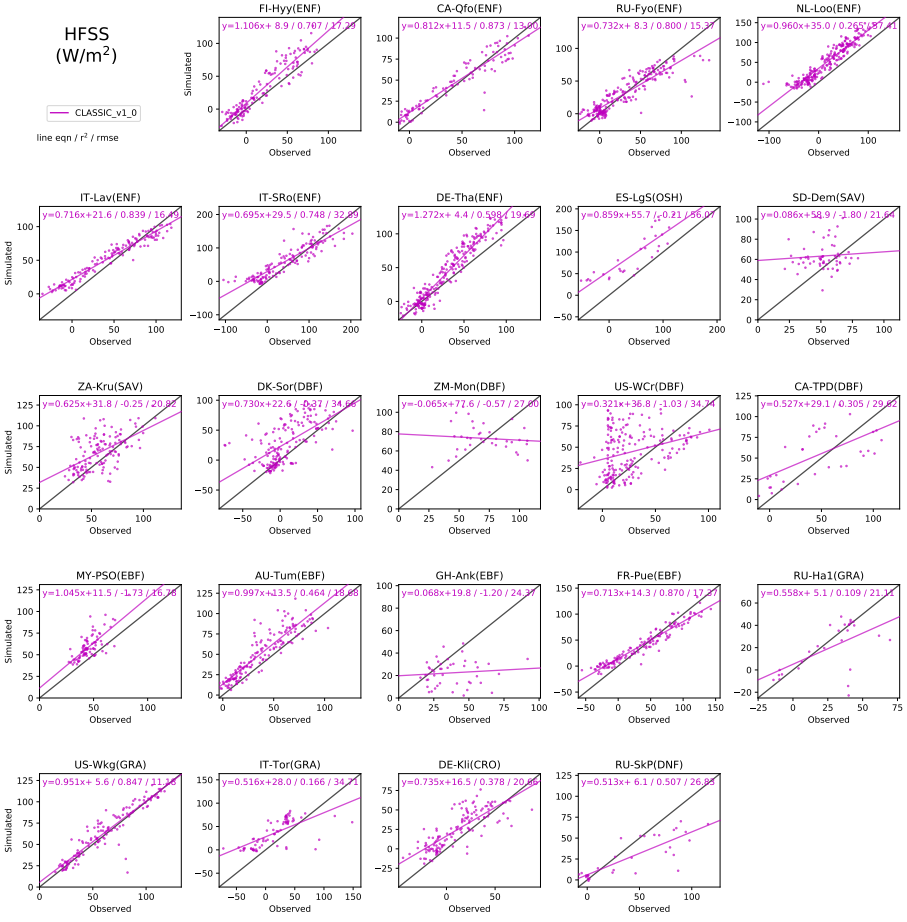

Figure S9. Scatter plot of simulated and observed HFSS for 23 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

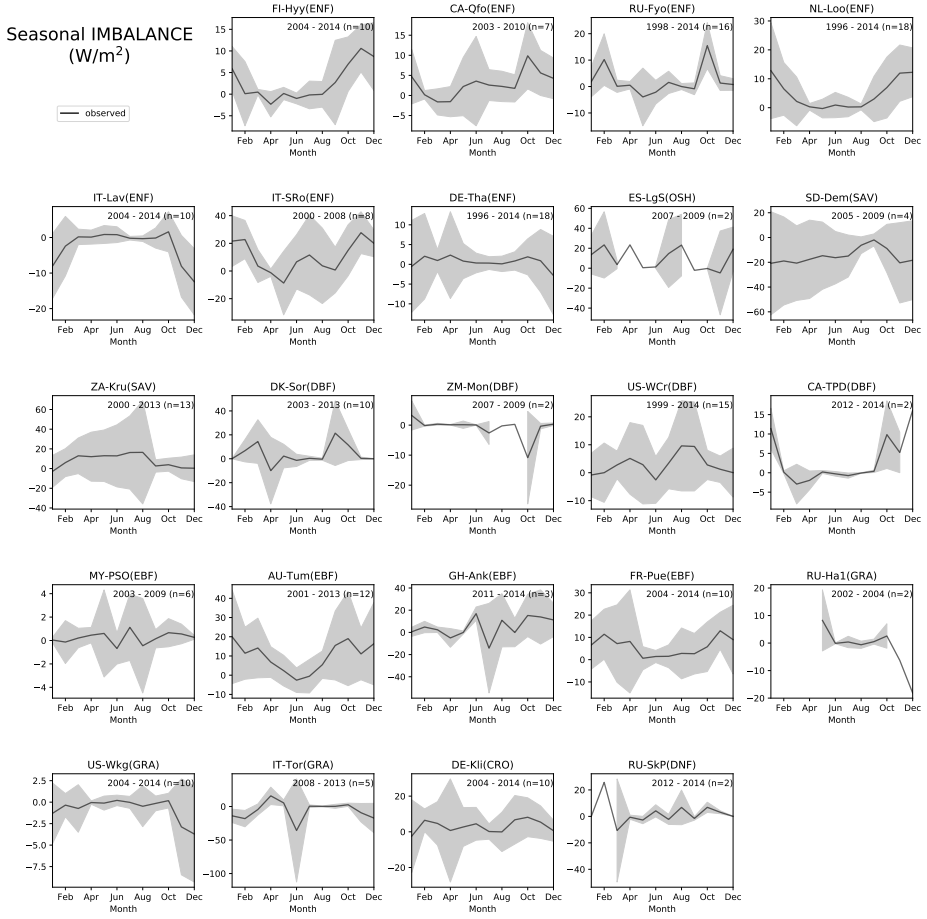

Figure S10. The mean seasonal cycle of imbalance for 23 FLUXNET sites as derived from the EC tower observations. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). See discussion in main text.

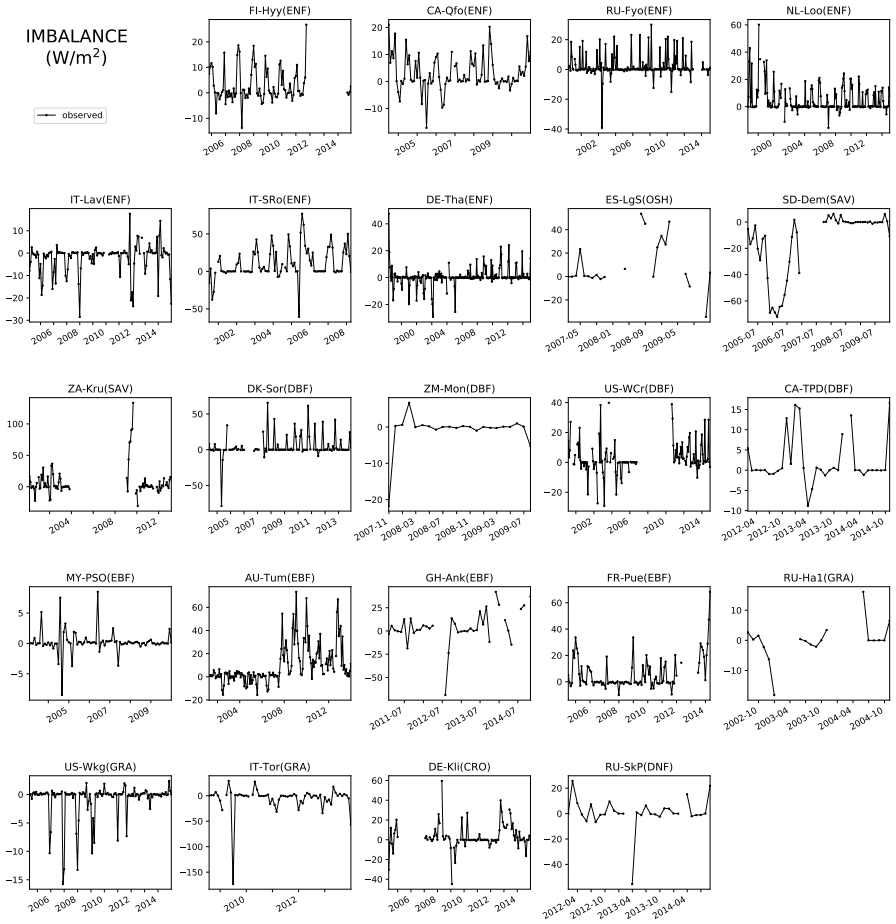

Figure S11. Timeseries of imbalance for 23 FLUXNET sites as derived from the EC tower observations. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). See discussion in main text.

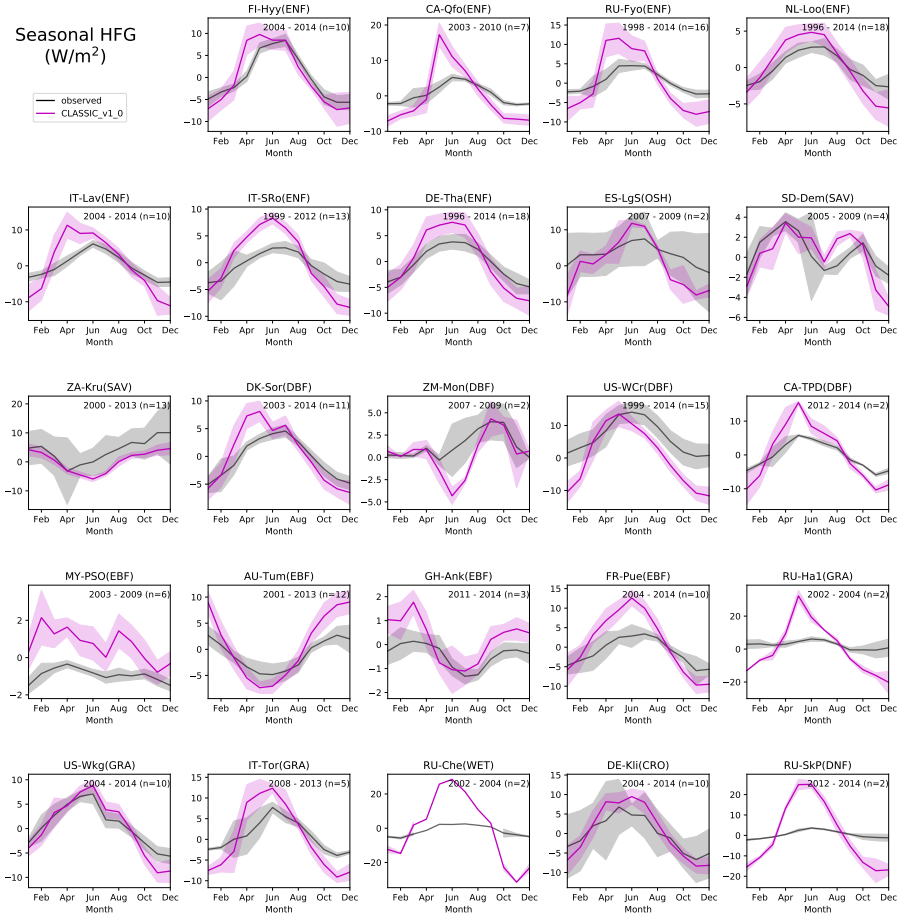

Figure S12. The mean seasonal cycle of HFG for 24 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

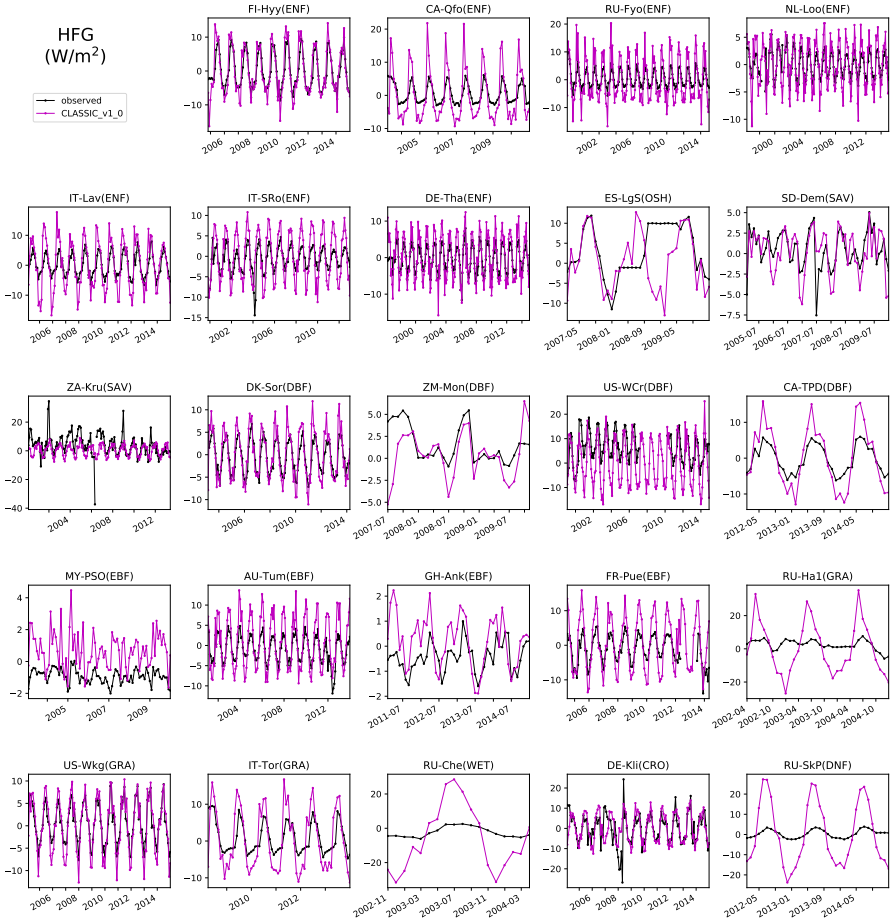

Figure S13. Timeseries of HFG for 24 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

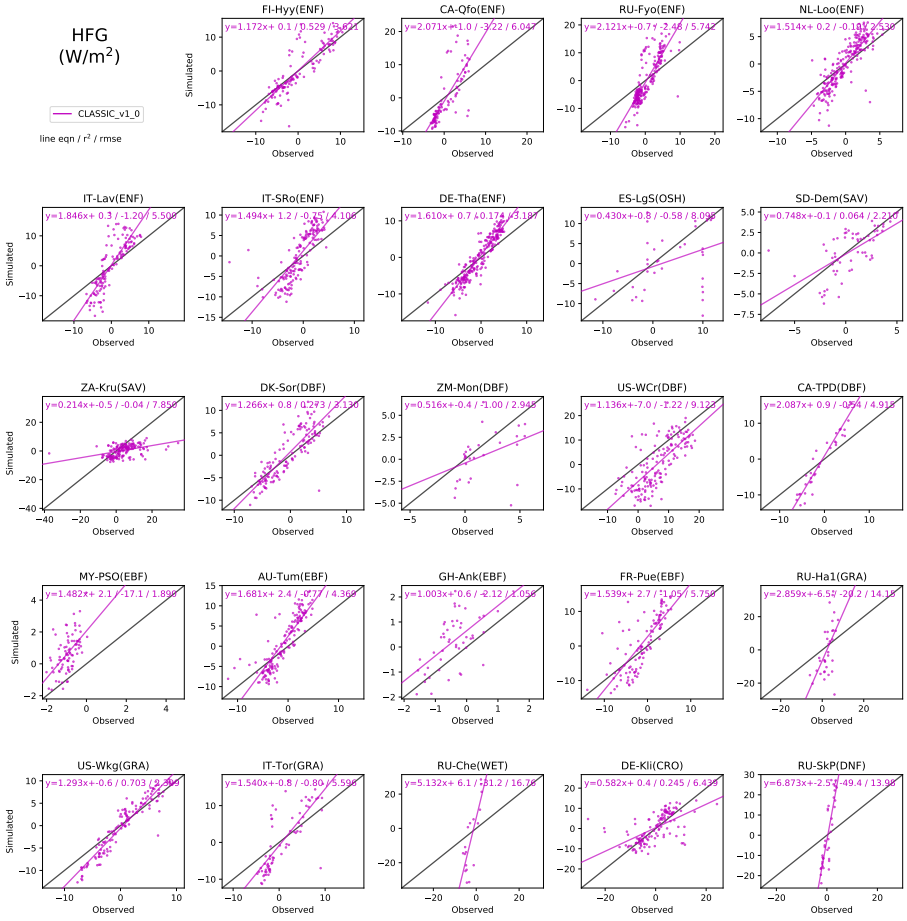

Figure S14. Scatter plot of simulated and observed HFG for 24 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

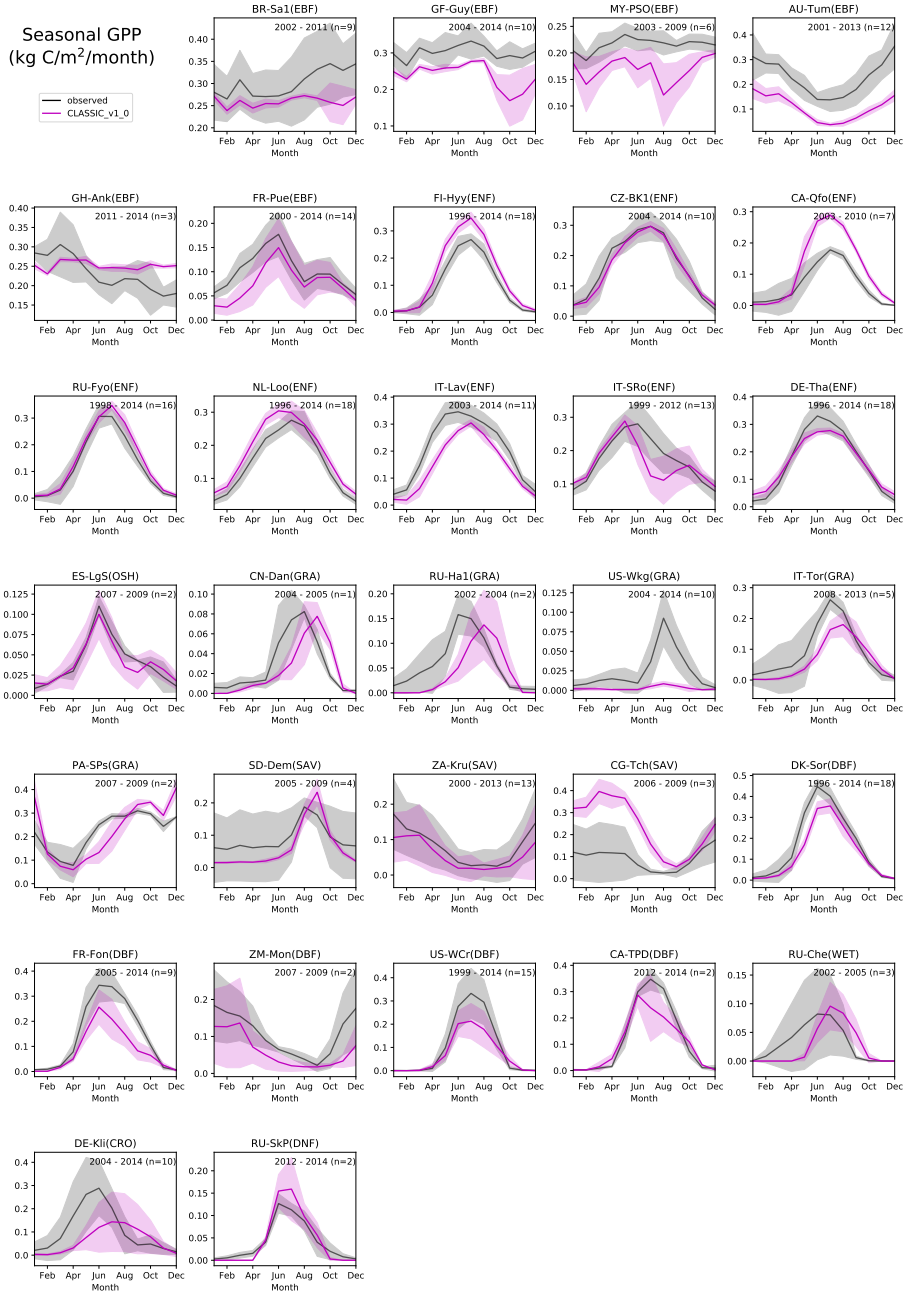

Figure S15. The mean seasonal cycle of GPP for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

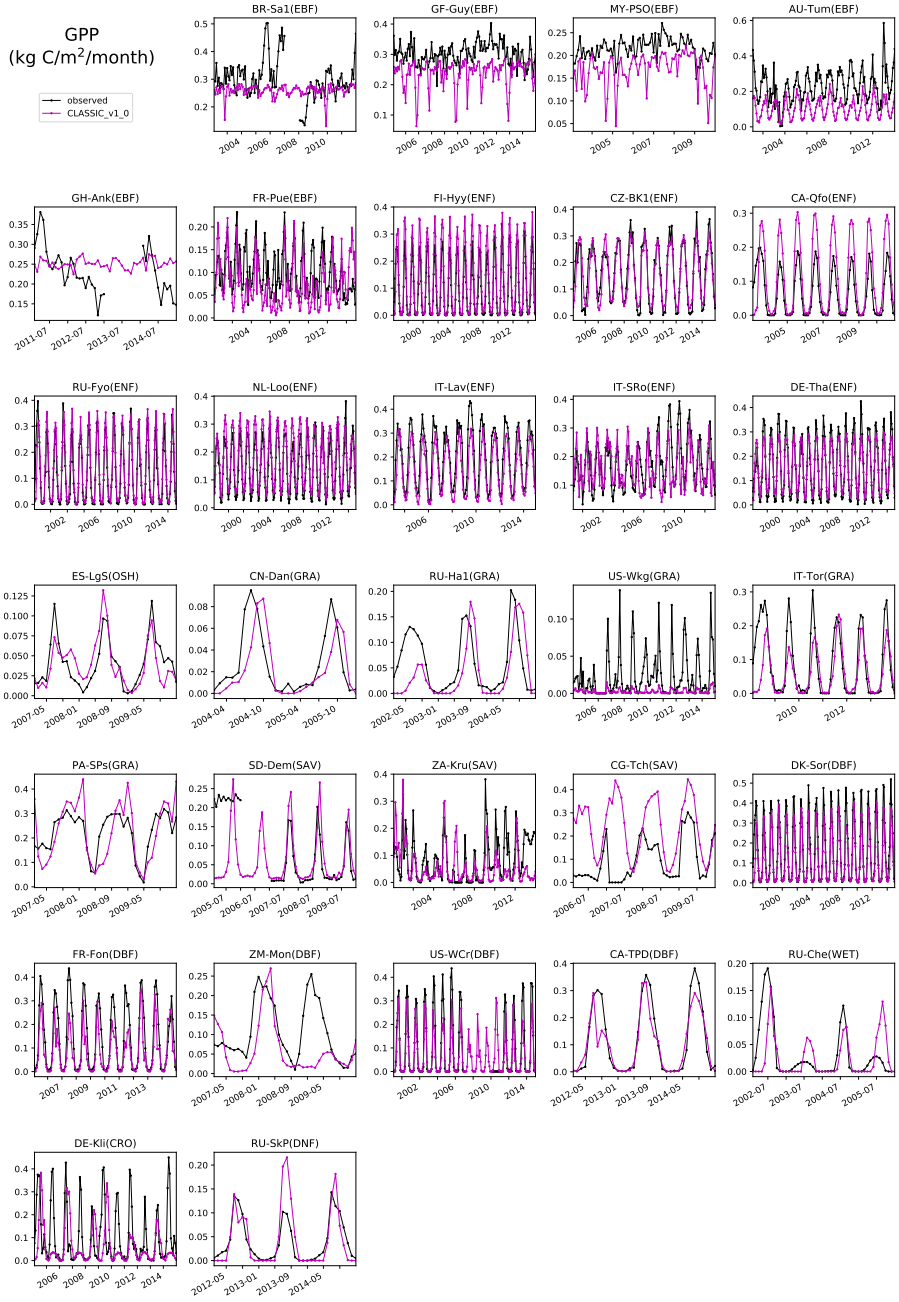

Figure S16. Timeseries of GPP for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

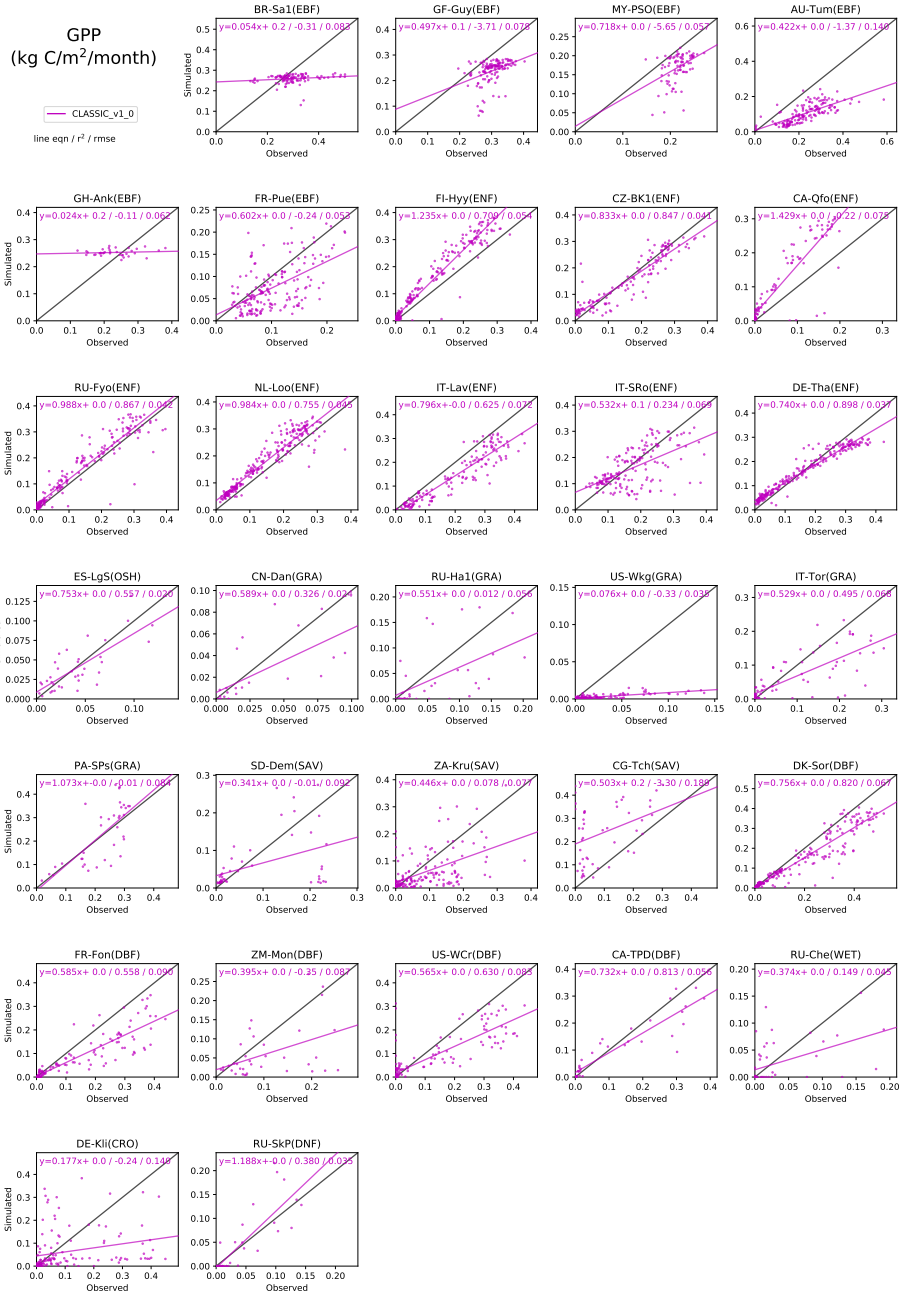

Figure S17. Scatter plot of simulated and observed GPP for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

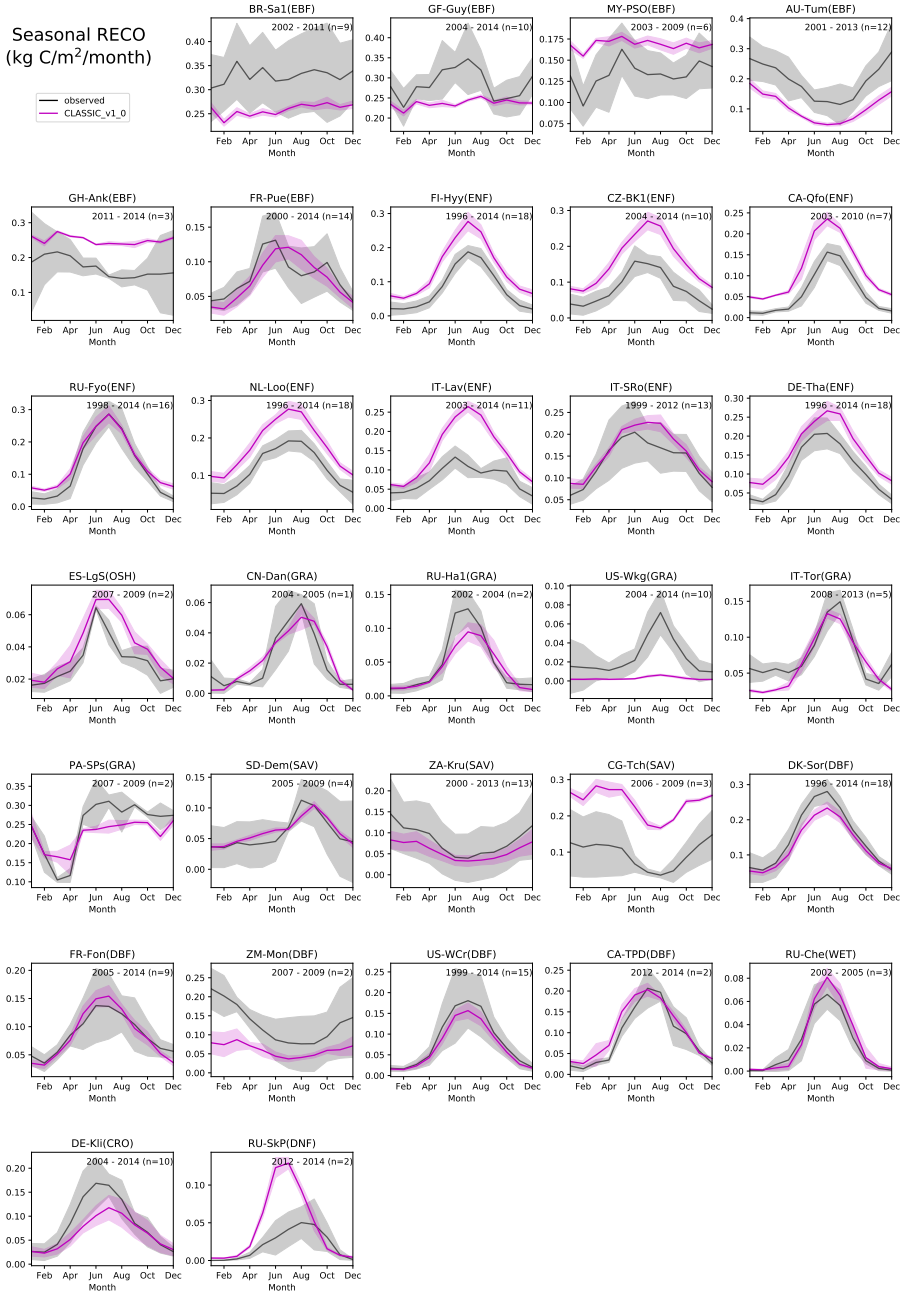

Figure S18. The mean seasonal cycle of RECO for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

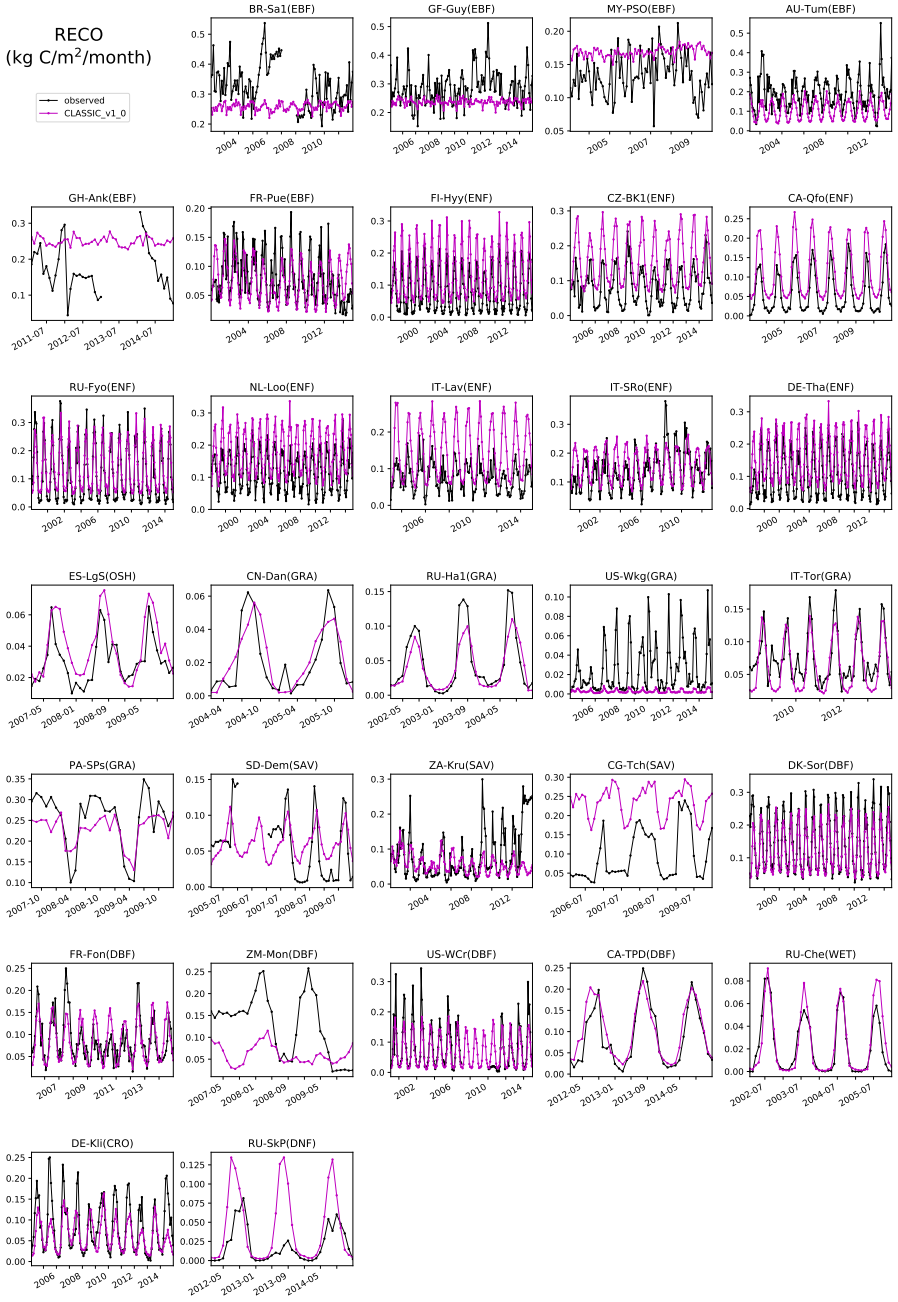

Figure S19. Timeseries of RECO for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

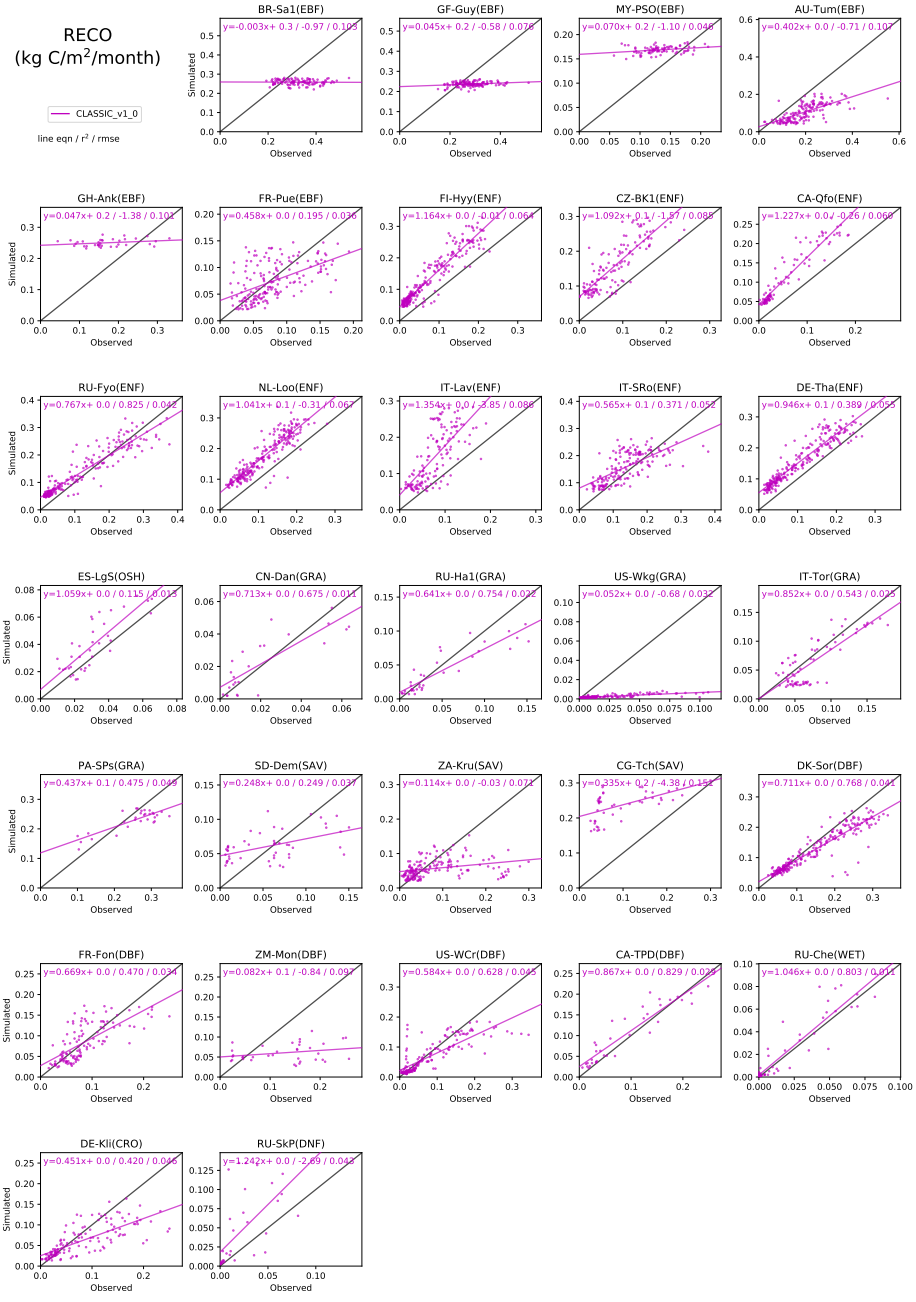

Figure S20. Scatter plot of simulated and observed RECO for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

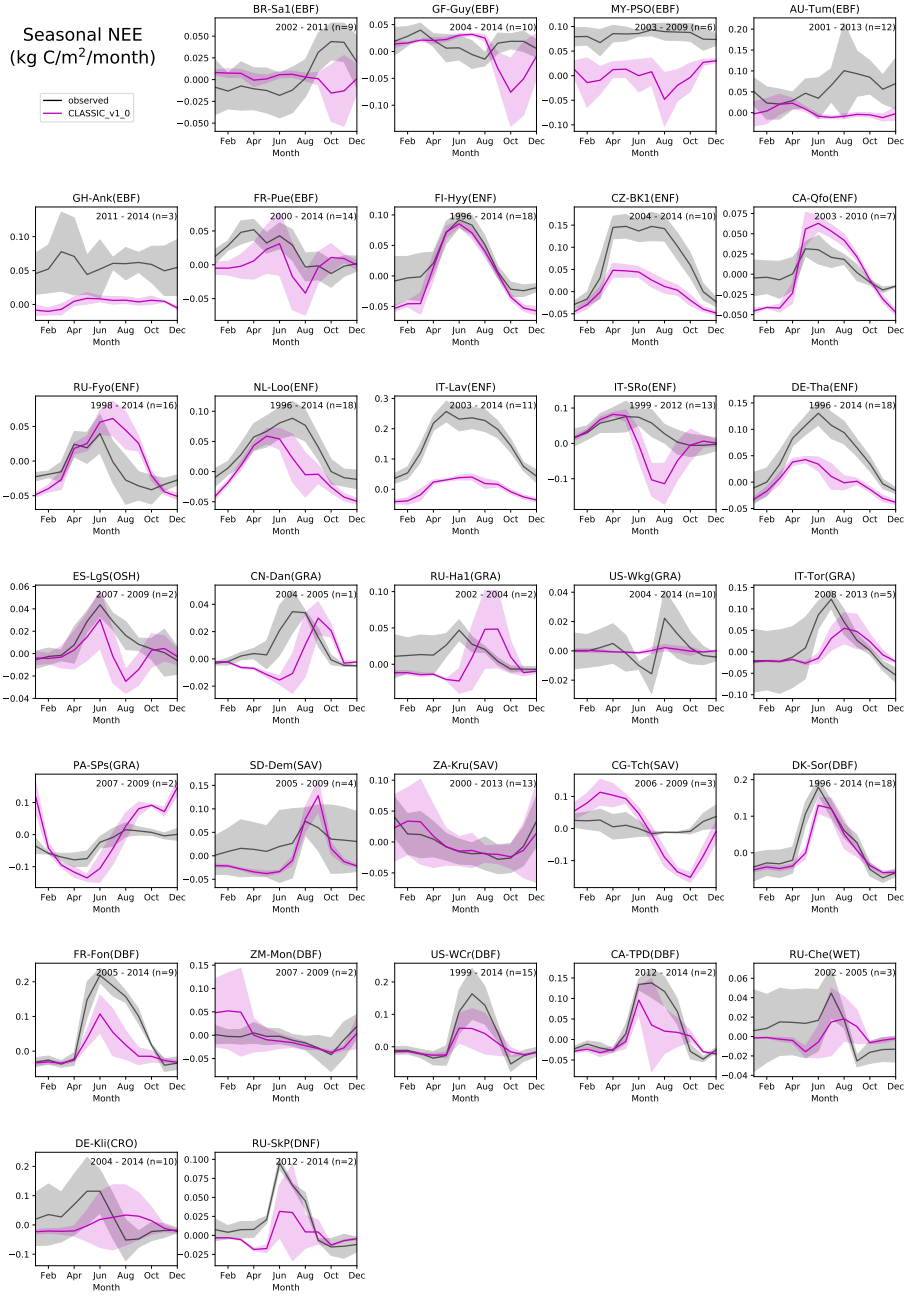

Figure S21. The mean seasonal cycle of NEE for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

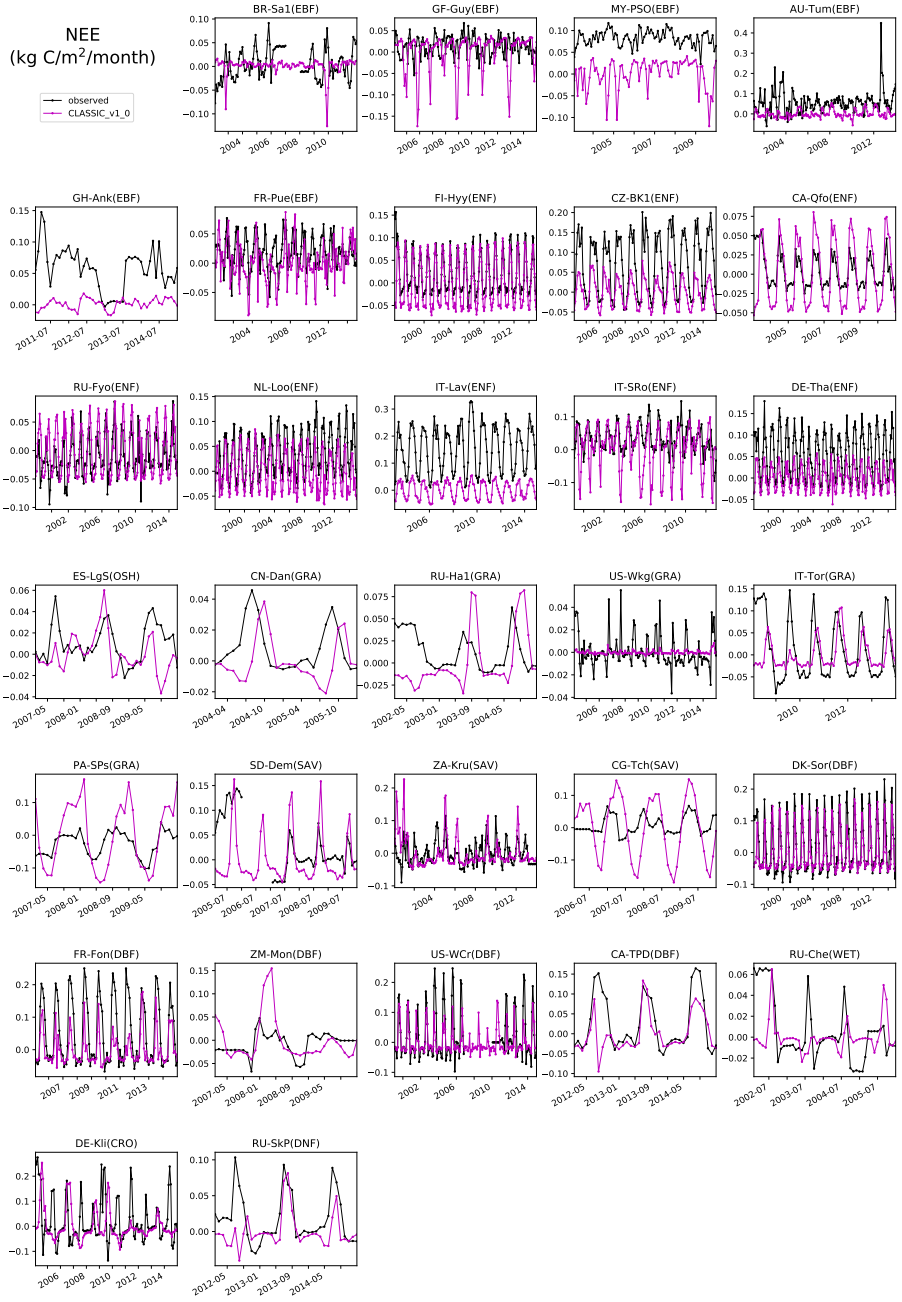

Figure S22. Timeseries of NEE for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text).

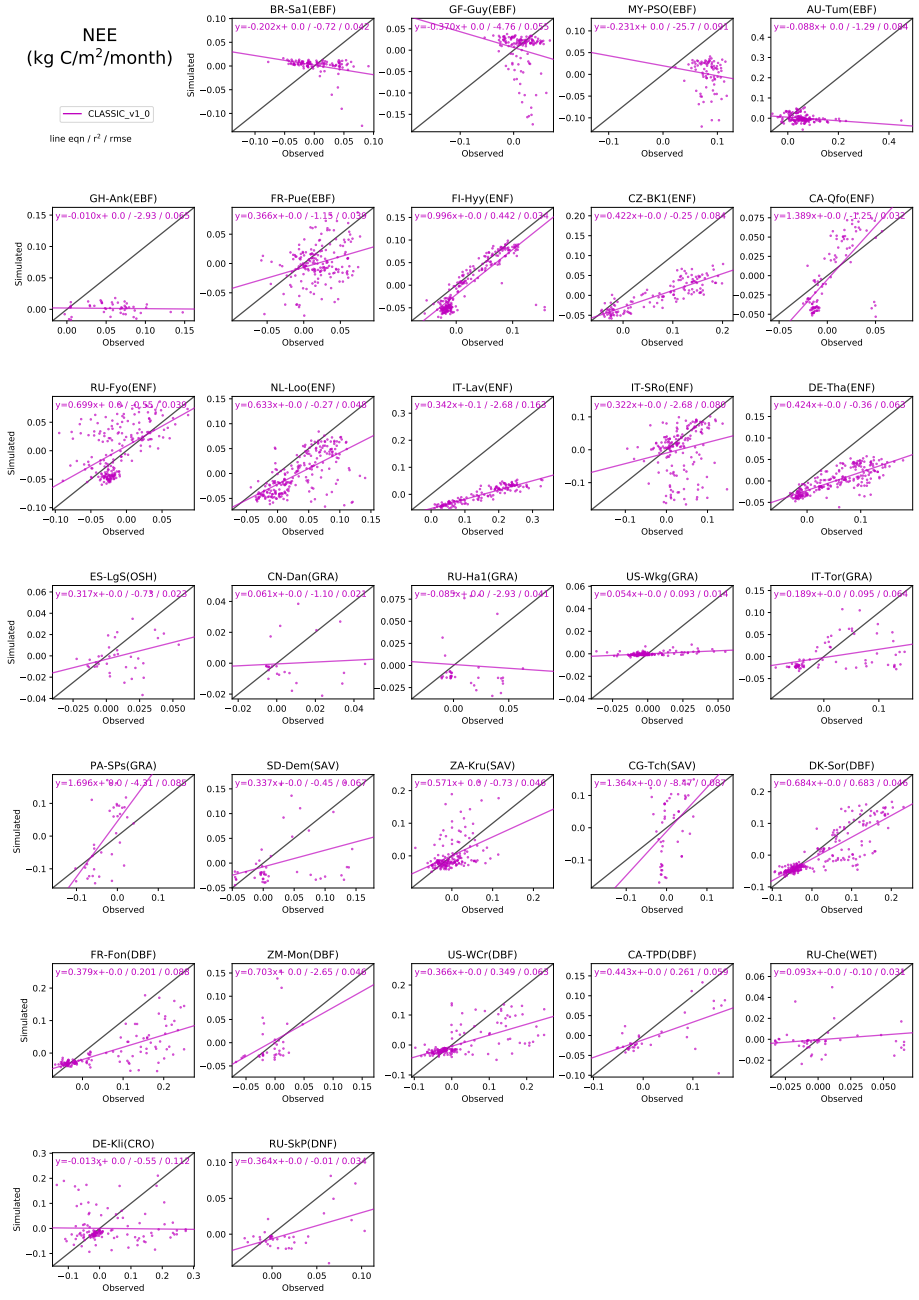

Figure S23. Scatter plot of simulated and observed NEE for 31 FLUXNET sites as simulated by CLASSIC. IGBP biome of each site is listed alongside its FLUXNET name (see Table 1 in main text). Equation of line of best, coefficient of determination, and RMSE (see main text) are included for each site.

2 CLASSIC coding conventions

This outlines the coding conventions adhered to in the CLASSIC source code. Some rules are marked as *(required)*, and should be followed at all times unless it is breaking to the code. Other rules are *(suggested)*, meaning it is a good idea to follow them, but failing to do so will not necessarily be grounds for rejecting the code from integration.

2.1 Language and Units

- Program components and comments should be written in simple English unless there exists a very good reason for not doing so *(required)*
  - Program components include variables, functions, subroutines, modules, filenames, and other constructs
  - Always bear in mind that your code may be read by people who are not proficient English speakers
- Variables, subroutines, functions, and modules should have names which clearly represent their intended functions *(required)*
  - Single-letter variables are acceptable only for transient use where the code is easily readable
- Units must be clearly stated in comments *(required)*
  - SI units are prefered unless there is a good reason not to use them
  - Standard SI prefixes may be used

2.1.1 Case conventions

- Case conventions vary by project, and contributors are encouraged to follow whatever case conventions are used. CLASSIC adopts camelCase for most of its subroutine and variable naming, so contributors are expected to do the same *(required)*

2.2 Free-Form Fortran

- Free-form Fortran 90/95/08 is the standard for this codebase; source code files should have a suffix of .f90 or .F90 *(required)*
  - F90 is more readable, easier to maintain, and boasts native support for more functionality
  - Fixed-form syntax contains many obsolete features and is more difficult to read

2.3 General Structure

- Modules should always have their own file *(required)*
- Related functions, subroutines, and variables should be grouped together into modules with descriptive names *(suggested)*
- Modules, functions, and subroutines should always end with its own name to help delineate the extent of that construct within the source code *(required)*

```
module foo

...
```

```
end module foo

subroutine bar( )

...

end subroutine bar

integer function moo( )

...

end function moo
```

2.3.1 Whitespace

- Two-space indentation blocks will be used for this codebase *(required)*
- Tabs are not to be used in place of spaces
- Block comments should follow the same whitespace guideline as the code block they are contained within *(required)*
- Trailing whitespace should be eliminated from every line *(required)*

```
do i = 1, 50
  if (MODULO(i, 5) == 0) then
    call someRoutine(i) ! this is a comment block
                        ! using the same whitespace guideline as
                        ! the code block in which it is contained
  end if
end do
```

2.3.2 Alignment

- Effort should be made to align the attributes of code segments if readability will be improved *(suggested)*
- Only do this if it won't add excessive whitespace

```
real, intent(in) :: inputArgument
real, intent(inout) :: inputOutputArgument
real, intent(out) :: outputArgument

rmlmossac_t   => ctem_tile%rmlmossac_t
gppmossac_t   => ctem_tile%gppmossac_t
litrmsmossrow => vrot%litrmsmoss
Cmossmasrow   => vrot%Cmossmas
dayl_maxrow   => vrot%dayl_max
sdeprot       => class_rot%sdeprot
RADJROW       => class_rot%RADJROW
```

2.3.3 Line Length and Continuation

- Maximum line length should be 120 characters including indentation *(suggested)*
- This increases readability of code, and applies to comments as well
- Continuation lines will have an ampersand at the end of the line to be continued *(required)*
- A space should be present before the ampersand, and this should all be contained within the suggested 120 character limit
- The following line will be indented by additional whitespace at the discretion of the developer to maintain a clean look (see below)

```
character(50) :: person_one, person_two, person_three, &
                                 person_four
person_one = ``Ada Lovelace''
person_three = ``Alan Turing''
```

2.4 Variable Declaration and Attributes

2.4.1 Variable Description

- Variables should be described using either inline comments for each variable, or with a comment block describing a group of variables *(suggested)*
- Comments should be Doxygen readable, with LaTeX use where appropriate to allow for proper Doxygen rendering *(required)*

```
real, intent(out) :: RHOSROT(NL, NM) !< Density of snow \f\([kg m^{-3} ]\f\)
real, intent(out) :: SNOROT (NL, NM) !< Mass of snow pack \f\([kg m^{-2} ]\f\)
```

2.4.2 Implicit None

- ‘implicit none‘ must be declared in **every** program component *(required)*
- This prevents implicit variable declarations, a huge source of bugs

2.4.3 Attributes

- Variable declarations must use a double colon ‘::‘ even if no other attributes are present *(required)*
- Bad: ‘integer example_variable‘
- Good: ‘integer :: example_variable‘
- If multiple declarations occur with the same attributes, the attributes should be listed in the same order *(required)*

```
integer, allocatable, pointer :: foo
integer, allocatable, pointer, dimension(:,:) :: bar
```

2.4.4 Subroutine Variables

- Variables used within a subroutine should have the ‘intent()‘ attribute matching their intended use *(required)*
- More information on subroutines and intent can be found at https://en.wikibooks.org/wiki/Fortran/Fortran_procedures_and_functions.
- Whenever possible, arguments passed into a subroutine should have names matching those found within the subroutine *(suggested)*
- This applies to function calls as well

```
subroutine square_cube (i, i_squared, i_cubed)
  implicit none
  integer, intent(in)  :: i                    ! input
  integer, intent(out) :: i_squared, i_cubed   ! output
  i_squared = i ** 2
  i_cubed   = i ** 3
end subroutine square_cube

program xx
  implicit none
  integer :: i, i_squared, i_cubed
  i = 4
  call square_cube (i, i_squared, i_cubed)
  print *, "i, i^2, i^3 = ", i, i_squared, i_cubed
end program xx
```

2.5 Fortran Modules

- Avoid using module level variables *(suggested)*
- Always specify ‘use, only‘ when importing from another module *(required)*
- This helps developers trace where variables, functions, and subroutines originate and makes the organization of the code much clearer and helps reduce clutter in the namespace.

```
module foo
  use bar, only : fun1, fun2
  ...
end module foo
```

- Entities within a module should be set to private by default, and declared public only when they are intended to be accessed from outside the module

```
module exampleModule
  implicit none
  ...
  private
```

```
    ! Sets default to private

    ...
    integer, parameter :: alpha, beta, delta, gamma
    ...

    public  :: alpha, beta
    ! Sets the specified entities to public

    contains
      ...
end module exampleModule
```

2.6 Operators

2.6.1 Array arithmetic

- Explicit indexing will be used for array arithmetic *(required)*
- This prevents undefined behaviour caused by unintentional operations on array halos

  Example (where A, B, C are 2-dimensional arrays)

  Bad:

```
C = A*B
```

  Good:

```
C(i,j:i,k) = A(i,j:i,k) * B(i,j:i,k)
```

2.6.2 Assignment and Arithmetic Operators

- Assignment and arithmetic operators should maintain a space on either side *(suggested)*
- This improves interpretability of arithmetic operations and assignments

```
x = ((15 - y) / (13 * z)) ** 3
```

2.6.3 Relational and Logical Operators

- The new standard relational operators will be used, as shown below *(suggested)*
- These apply to comparable data types such as ‘real‘ or ‘character‘, and prevent accidental mixing with logical types
- Logical variables and operations will continue to appear similar to the old relational syntax

  ‘.true. .false.‘ variables and ‘.eqv. .neqv. .not. .and. .or.‘ operators
- Logical evaluations should not use redundant operations as shown below: *(suggested)*

Table S8. Relational operators

Old (deprecated) syntax	New syntax
.eq.	==
.ne.	/=
.lt.	<
.le.	<=
.gt.	>
.ge.	>=

```
Bad:

doProc = .true.
...
if (doProc .eqv. .true.) then
  ...
end if

if (doProc .neqv. .true.) then
  ...
end if

Good:

doProc = .true.
...
if (doProc) then
  ...
end if

if (.not. doProc) then
  ...
end if
```

2.7 Control Flow

2.7.1 Loops

- Loops will terminate with ‘end do‘ and not ‘continue ____‘ *(required)*
- Each loop must terminate with its own ‘end do‘ (no sharing loop terminators)
- This is an F90 standard to promote comprehension
- Loops will not be labelled with numbers as was common in F77 *(required)*

```
do i = 1, 10
  ! logic happens in here
  ...
end do
```

– In the case of nested or overlapping logic constructs, **descriptive** labelling may be used to distinguish these constructs *(suggested)*

```
outer_loop: do i = 1, 10
  ...
  inner_loop: do j = 1, 20
    ...
  end do inner_loop
  ...
end do outer_loop
```

2.7.2 If-Else Statements

– Positive logic should be used in if-evaluations when the contained code blocks are of comparable length *(suggested)*

– Positive logic refers to checking that a variable is true rather than false

– An example is given below:

```
if (exampleVariable /= 5) then
  call routineX()
else
  call routineY()
end if
```

Note the negative logic expression in the if test. This is easily refactored:

```
if (exampleVariable == 5) then
  call routineY()
else
  call routineX()
end if
```

– Include a space after the 'if' and before starting the logical expression *(suggested)*

– Group compound logical expressions using parentheses *(required)*

```
if ( (a == b) .and. (b == c) ) then
  ...
end if
```

2.7.3 Goto

– Goto statements are to be avoided whenever possible *(required)*

– If absolutely necessary, goto statements should be thoroughly documented *(required)*

– Including the reason for their use

2.8 Commenting and Documentation

- Liberal use of good comments is encouraged
- See Language and Units above for language guidelines
- Inline comments should be descriptive and concise
- Longer documentation should be placed at the end of the file
- Changelogs are not necessary due to the use of a Version Control System (VCS) *(required)*
- VCSs like git tracks users and changes automatically
- Doxygen is the documentation tool used in this repository. Comments should conform to Doxygen's Fortran standards as outlined at http://www.doxygen.nl/manual/docblocks.html#fortranblocks

  The subroutine example from above has been commented using Doxygen's specifications here:

```
!> Computes the square and cube of an input value, and
!! writes them to the output values
!! @param i input value
!! @todo Shame users for passing in negative numbers
subroutine square_cube (i, i_square, i_cube)
  implicit none
  integer, intent(in)  :: i                  !< Our input variable
  integer, intent(out) :: i_square, i_cube   !< Our output variables
  i_square = i ** 2
  i_cube   = i ** 3
end subroutine square_cube
```

2.9 Acknowledgements

Several coding conventions were used as a guide for developing this document including the Joint UK Land Environment Simulator (JULES) Coding Standards, NASA Goddard ModelE Coding Conventions, and the GEM-MACH Coding Standards.