Kamodo Analysis Suite

A Functional API for Space Weather Models and Data

Asher Pembroke
Lutz Rastaetter
Darren Dezeeuw
Katherine Garcia-Sage

SERVING THE SPACE WEATHER COMMUNITY

Mission Statement: To enable, support, and perform research for next generation space science and operational space weather models through an interagency partnership.

COMMUNITY COORDINATED MODELING CENTER

SERVING THE SPACE WEATHER COMMUNITY

Mission Statement: To enable, support, and perform research for next generation space science and operational space weather models through an interagency partnership.

A few of the models hosted at CCMC

SOLAR

PFSS
J. LUHMAN
@UC, BERKELEY

WSA-PF+CS
N. ARGE @AFRL

ANMHD
B. ABBET, ET AL
@BERKELEY

MAS
LINKER ET AL
@SAIC, CA

HELIOSPHERE

ENLIL, ENLIL+CONE
D. ODSTRCIL
@ UC BOULDER

BATSRUS
GOMBOSI, TOTH ET AL
@ CSEM, U. MICHIGAN

HELIOSPHERIC
TOMOGRAPHY + IPS/SMEI
JACKSON, HICK
@CASS, UCSD

EXOSPHERIC SOLAR WIND
H. LAMY, V. PIERRARD
@IASB-BIRA

GLOBAL MAGNETOSPHERE

LANL* NEURAL NETWORK
YIQUN, KOLLER
@LANL

OPEN GGCM
RAEDER, FULLER-ROWELL
@SSC, UNH

BATSRUS
GOMBOSI, TOTH ET AL
@ CSEM, U. MICHIGAN

GUMICS
JANHUNEN
@FINISH METEROLOGICAL INST

WINDMI
HORTON ET AL
@ U TEXAS

CMIT LFM-MIX
LYON ET AL
@DARTMOUTH, NCAR-HAO,
JHU-APL,CISM

THERMOSPHERE/IONOSPHERE

SAMI-3
HUBA ET AL
@NRL, ICARUS
RES. INC.

USU-GAIM
SCHUNK ET AL
@UTAH STATE

GITM
RIDLEY

PBMOD
RETTERER

CTIP-E
ROWELL ET AL
@CASS/USU

IRI
BILITZA
@NASA/GSFC

OVATION PRIME
NEWELL
@JHU APL

WEIMER MODEL
@VIRGINIA TECH

ABBYNORMAL
ECCLES ET AL
@CASS/USU

COSGROVE-PF
COSGROVE
@SRI INTL., CA

ATMOSPHERE:
MSISE
HEDIN

INNER MAGNETOSPHERE

LFM-RCM-MIX-TIEGCM
LYON, ET AL
@DARTMOUTH,NCAR/
HAO,APL,RICE

PLASMASPHERE
MODEL
V PIERRARD
@IASB-BIRA

FOK RADIATION
BELT ELECTRON
MEI-CHING FOK
@NASA/GSFC

BATSRUS

RICE CONVECTION
MODEL
WOLF,SAZYKIN
@RICE, TX

TSYGANENKO
TSYGANENKO
@ U ST-PETERSBURG

CIMI
FOK, BUZULUKOVA
@NASA/GSFC

BATSRUS

CRCM
MEI-CHING FOK
@NASA/GSFC

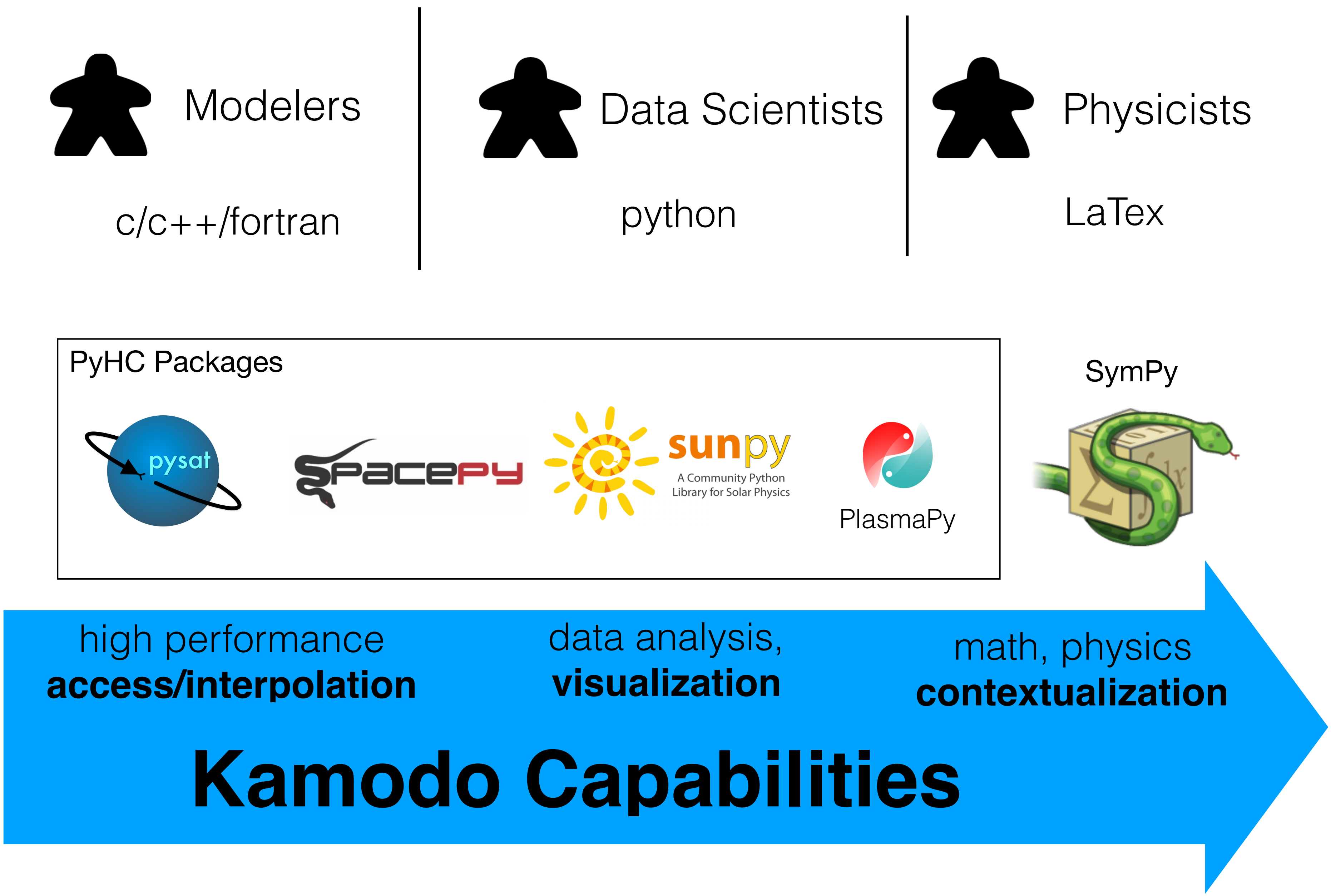
IGRF
MACMILLAN, MAUS
@IAGA

Kamodo Design Goals

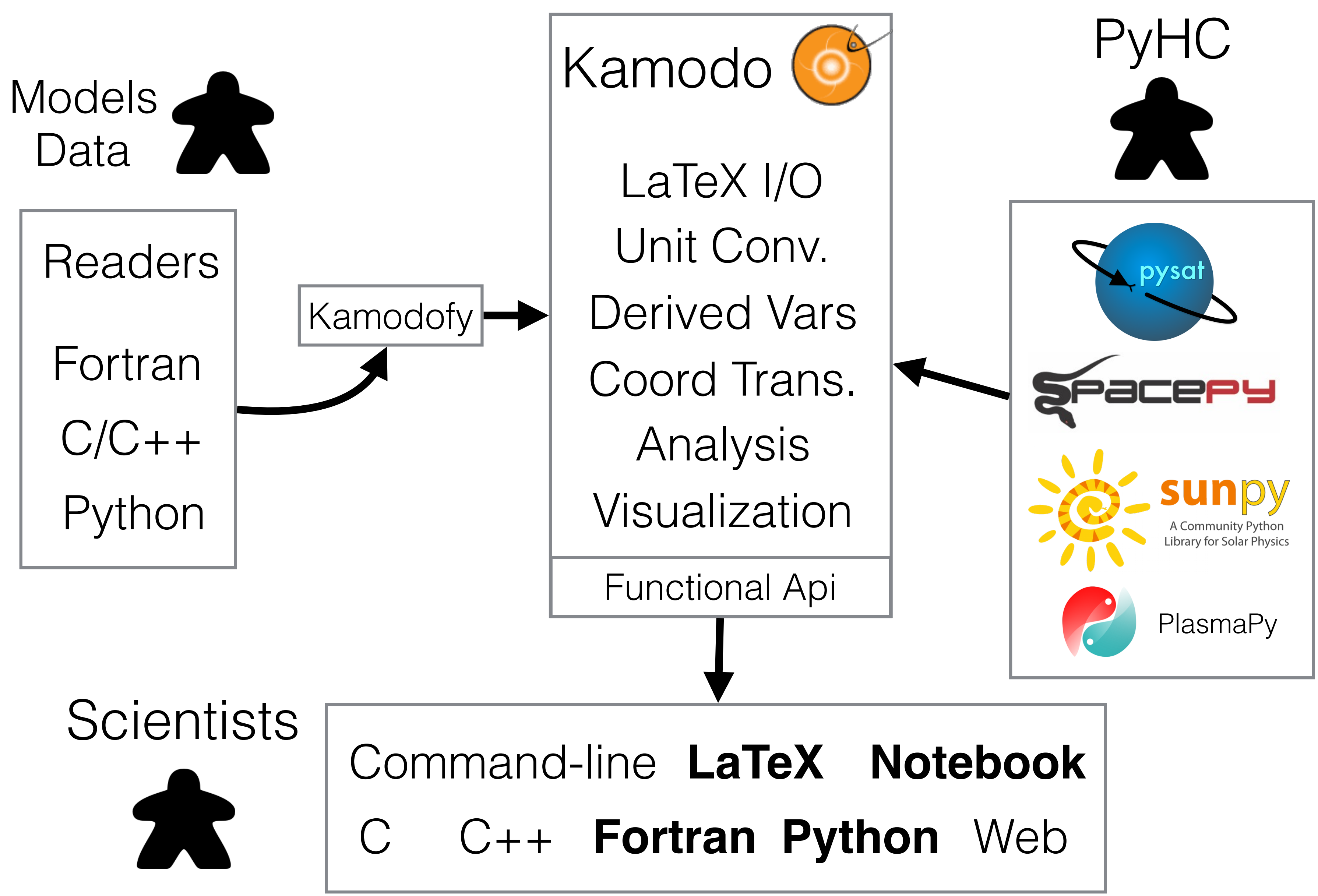
Endeavor to support a wide variety of users, models, and data sources:

- Quickly integrate new models and data
- LaTeX APIs for scientists and educators who don't code
- Model-agnostic interpolation API
- Format-agnostic
- Transparent, Permissive Metadata
- Automatic unit conversion
- Compatibility with helio-python ecosystem and support PyHC standards
- Instant visualization

Kamodo Architecture



Kamodo Architecture



Kamodo Functional API

Scientists work with models and data through Kamodo objects, which map symbols to interpolating functions or mathematical expressions.

```
kamodo = Kamodo('$e[J] = x^2$',
                'm[kg] = x',
                'v[km/s] = y',
               'p[kg*m/s] = m*v')

kamodo
```

$e(x)[J] = x^2$
$m(x)[kg] = x$
$v(y)[km/s] = y$
$p(x, y)[kg * m/s] = 1000m(x)v(y)$

```
kamodo.p(3,4)
```

```
12000
```

LaTeX-Formatted input

Function composition

Unit conversion

Functional interpolation api

Kamodo converts each expression to a highly optimized python function capable of operating on large arrays.

Kamodo Functional API

Existing Kamodo objects can be updated with new formulas using dictionary syntax. Function composition is applied automatically.

```
kamodo = Kamodo('$x = t^2$')
kamodo['g'] = 'y-1'
kamodo['f'] = 'g(x)'
kamodo
```

$x(t) = t^2$
$g(y) = y - 1$
$f(t) = g(x(t))$

Kamodo determines that f is a function of t through composition.

```
kamodo.f(3)
```

8

f(3) = 3^2 - 1 = 8

Many analysis and visualization problems can be framed in terms of *function composition*. Since most scientists are comfortable with function manipulation, this makes Kamodo ideal for their workflow.

@kamodofy

```python
from kamodo import kamodofy, Kamodo
import numpy as np

@kamodofy(units = 'kg/m^3', citation = 'Pembroke et. al, 2018')
def rho(x = np.array([3,4,5]), y = np.array([1,2,3])):
    """A function that computes density"""
    return x+y
```

Any python function can be "Kamodofied" using the @kamodofy decorator, adds metadata to a function.

@kamodofy

```python
from kamodo import kamodofy, Kamodo
import numpy as np

@kamodofy(units = 'kg/m^3', citation = 'Pembroke et. al, 2018')
def rho(x = np.array([3,4,5]), y = np.array([1,2,3])):
    """A function that computes density"""
    return x+y
```

Any python function can be “Kamodofied” using the @kamodofy decorator, adds metadata to a function.

```python
kamodo = Kamodo(rho = rho)
kamodo['den[g/cm^3]'] = 'rho'
kamodo
```

$\rho(x, y)[kg/m^3] = \lambda(x, y)$

$\operatorname{den}(x, y)[g/cm^3] = \frac{\rho(x, y)}{1000}$

Lambda indicates kamodofied function

@kamodofy

```python
from kamodo import kamodofy, Kamodo
import numpy as np

@kamodofy(units = 'kg/m^3', citation = 'Pembroke et. al, 2018')
def rho(x = np.array([3,4,5]), y = np.array([1,2,3])):
    """A function that computes density"""
    return x+y
```

Any python function can be “Kamodofied” using the @kamodofy decorator, adds metadata to a function.

```python
kamodo = Kamodo(rho = rho)
kamodo['den[g/cm^3]'] = 'rho'
kamodo
```

$\rho(x, y)[kg/m^3] = \lambda(x, y)$

$\mathrm{den}\,(x, y)[g/cm^3] = \frac{\rho(x, y)}{1000}$

Lambda indicates kamodofied function

```python
kamodo.den(3,4)
```

```
0.007
```

Function composition & unit conversion

@kamodofy

```python
from kamodo import kamodofy, Kamodo
import numpy as np

@kamodofy(units = 'kg/m^3', citation = 'Pembroke et. al, 2018')
def rho(x = np.array([3,4,5]), y = np.array([1,2,3])):
    """A function that computes density"""
    return x+y
```

Any python function can be “Kamodofied” using the @kamodofy decorator, adds metadata to a function.

```python
kamodo = Kamodo(rho = rho)
kamodo['den[g/cm^3]'] = 'rho'
kamodo
```

$$\rho(x, y)[kg/m^3] = \lambda(x, y)$$

$$\operatorname{den}(x, y)[g/cm^3] = \frac{\rho(x, y)}{1000}$$

Lambda indicates kamodofied function

```python
kamodo.den(3,4)
```

```
0.007
```

Function composition & unit conversion

```python
kamodo.rho.meta # PyHC standard
```

```
{'citation': 'Pembroke et. al, 2018', 'units': 'kg/m^3'}
```

```python
kamodo.rho.data # PyHC standard
```

```
array([4, 6, 8])
```

Python-in-Heliophysics Community Standards require a data attribute - we satisfy this by calling functions with their default arguments.

@kamodofy

```python
from kamodo import kamodofy, Kamodo
import numpy as np

@kamodofy(units = 'kg/m^3', citation = 'Pembroke et. al, 2018')
def rho(x = np.array([3,4,5]), y = np.array([1,2,3])):
    """A function that computes density"""
    return x+y
```

Any python function can be “Kamodofied” using the @kamodofy decorator, adds metadata to a function.

```python
kamodo = Kamodo(rho = rho)
kamodo['den[g/cm^3]'] = 'rho'
kamodo
```

$$\rho(x, y)[kg/m^3] = \lambda(x, y)$$
$$\operatorname{den}(x, y)[g/cm^3] = \frac{\rho(x, y)}{1000}$$

Lambda indicates kamodofied function

```python
kamodo.den(3,4)
```

```
0.007
```

Function composition & unit conversion

```python
kamodo.rho.meta # PyHC standard
```

```
{'citation': 'Pembroke et. al, 2018', 'units': 'kg/m^3'}
```

```python
kamodo.rho.data # PyHC standard
```

```
array([4, 6, 8])
```

Python-in-Heliophysics Community Standards require a data attribute - we satisfy this by calling functions with their default arguments.

```python
help(kamodo.rho)
```

```
Help on function rho in module __main__:

rho(x=array([3, 4, 5]), y=array([1, 2, 3]))
    A function that computes density
```

Supporting Documentation

```python
kamodo.detail()
```

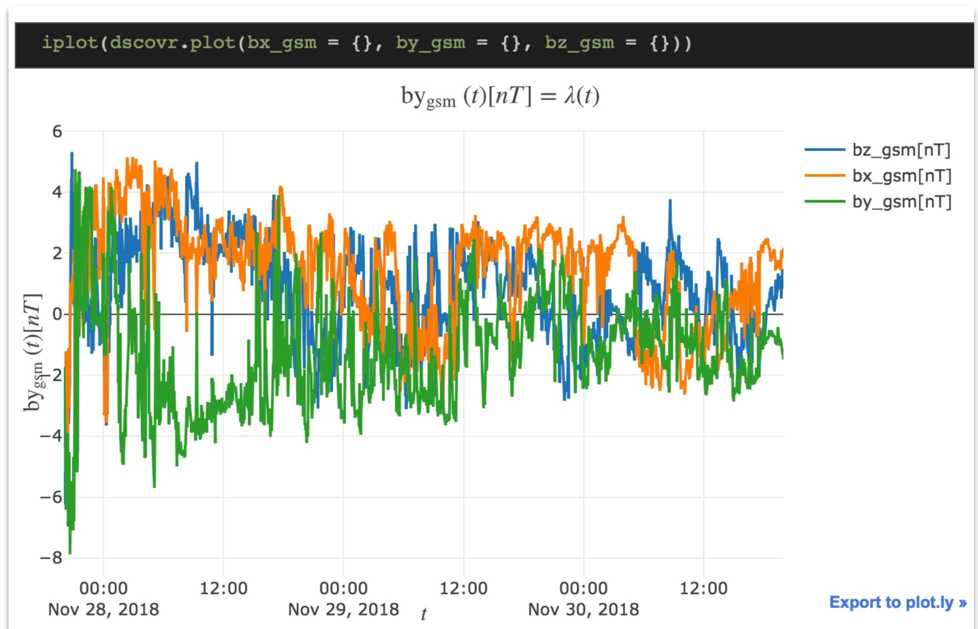
	lhs	rhs	symbol	units
rho(x, y)	rho	<function rho at 0x122bcde60>	rho(x, y)	kg/m^3
den(x, y)	den	rho(x, y)/1000	den(x, y)	g/cm^3

Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

1D Time Series

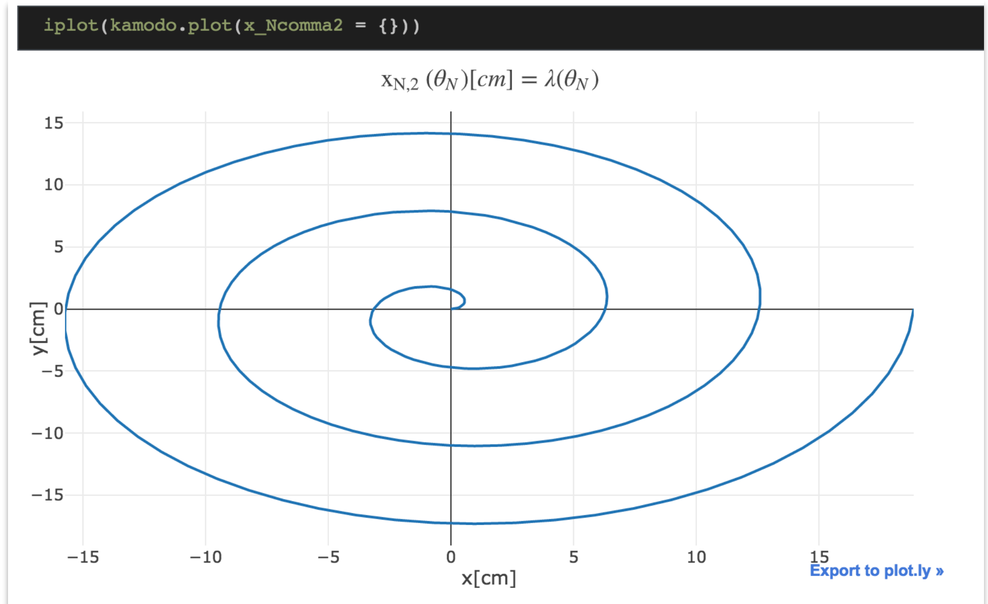


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

2D Parametric Curve

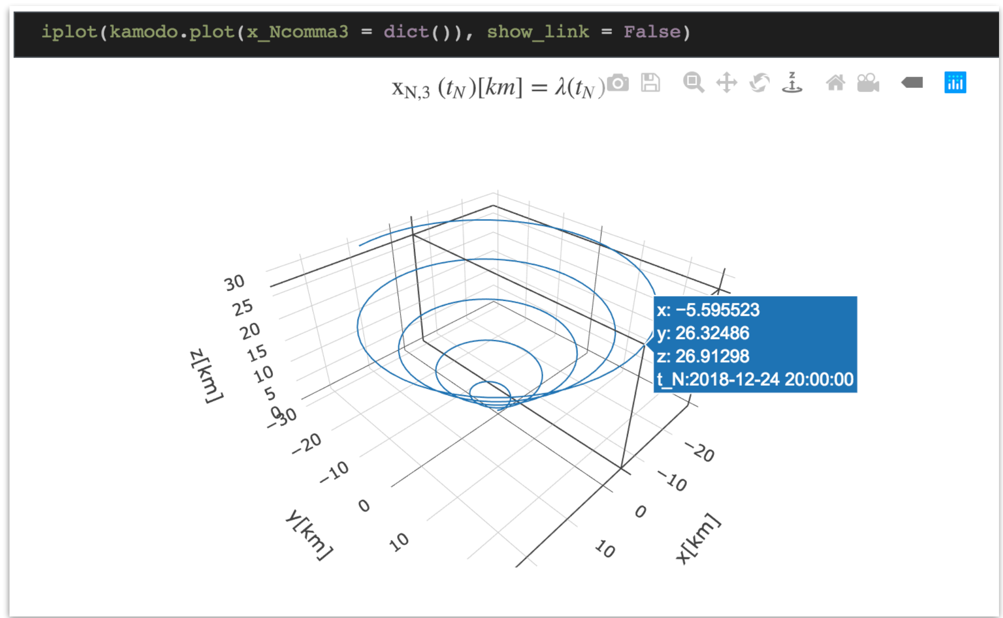


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

3D Parametric Curve

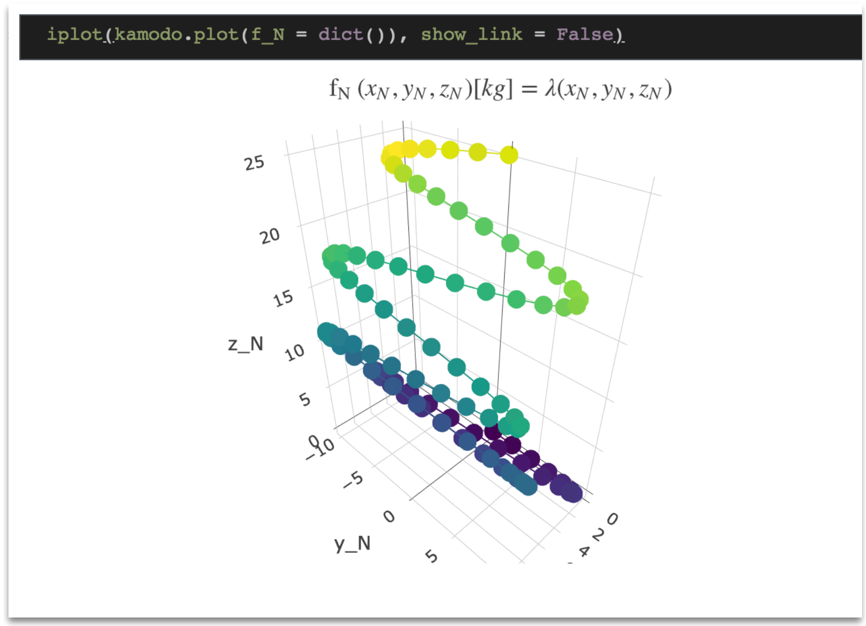


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Colored 3D curve

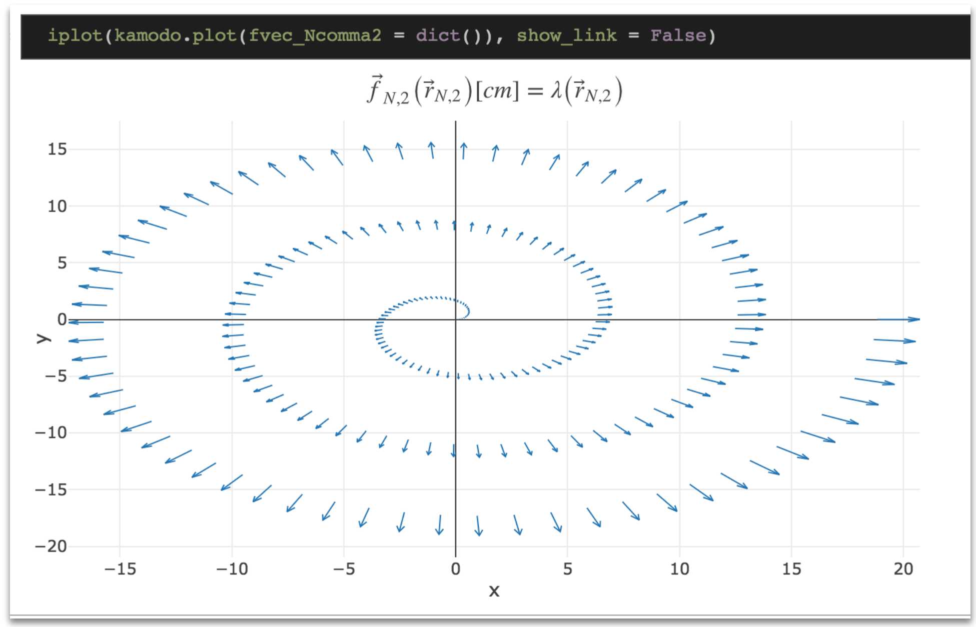


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

2-D Vector Curve

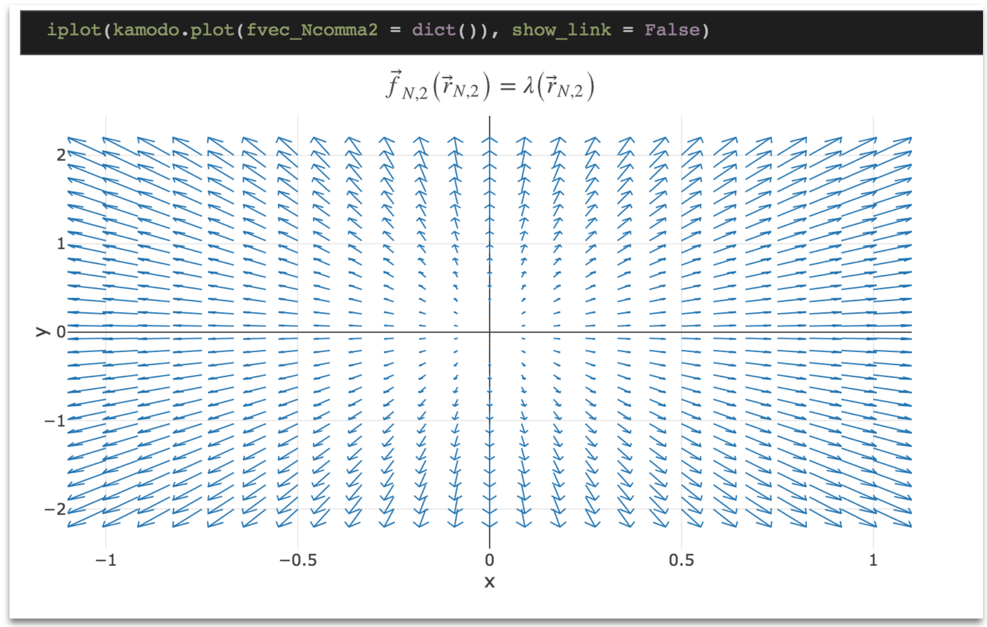


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

2-D Vector Field

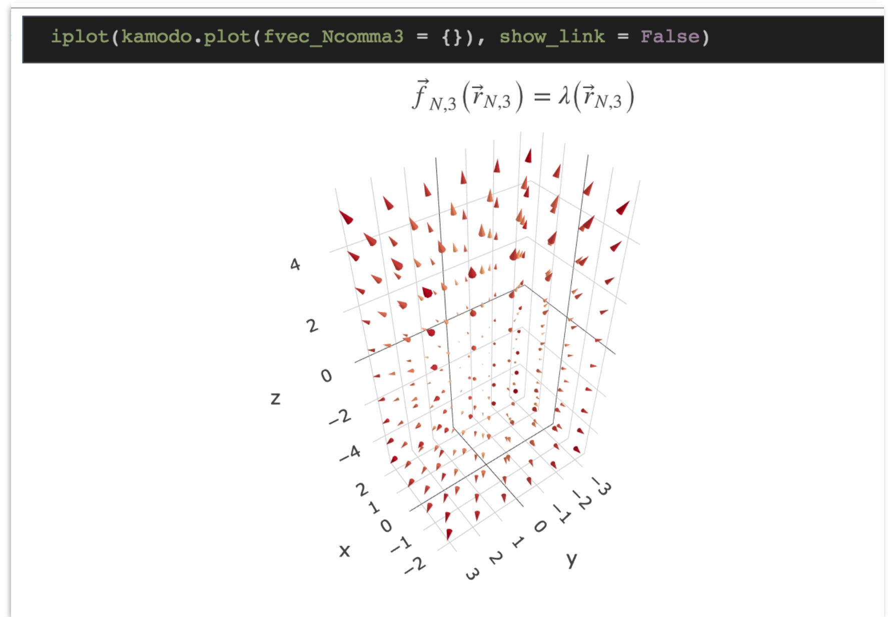


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

3-D Vector Field/Curve



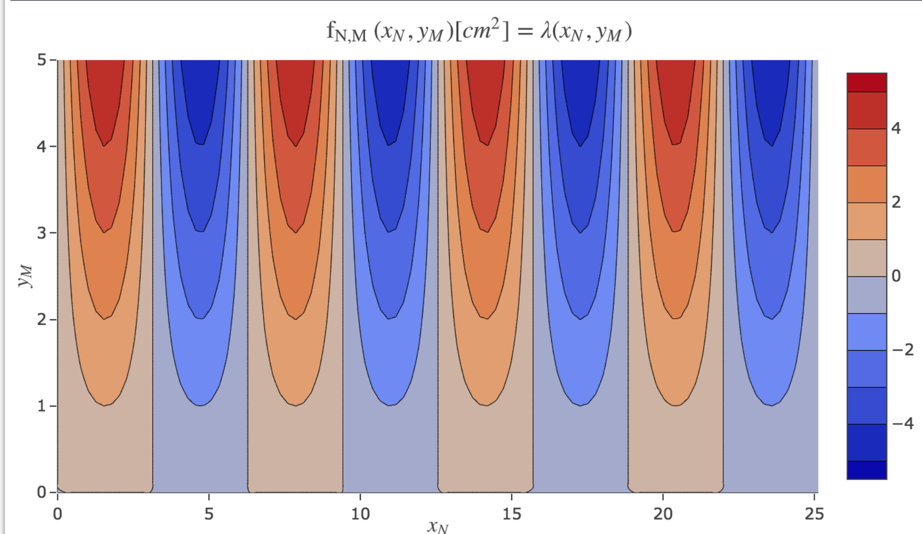
Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

nargs	arg shapes	out shape	Plot Type	notes
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Contour/Smooth

```
iplot(kamodo.plot(f_NcommaM = {}), show_link = False)
```

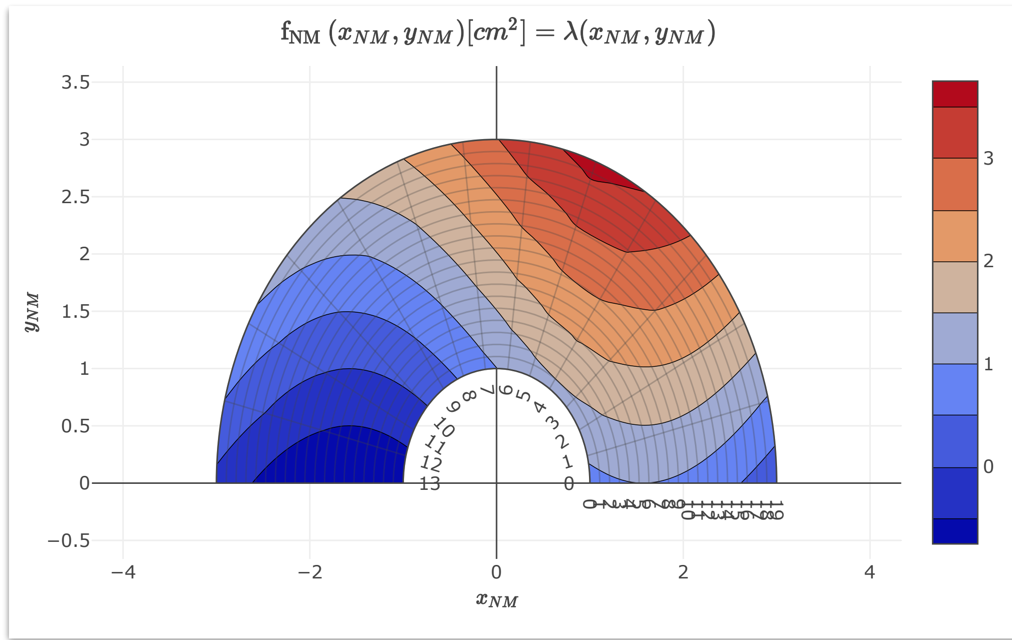


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Skew/Carpet



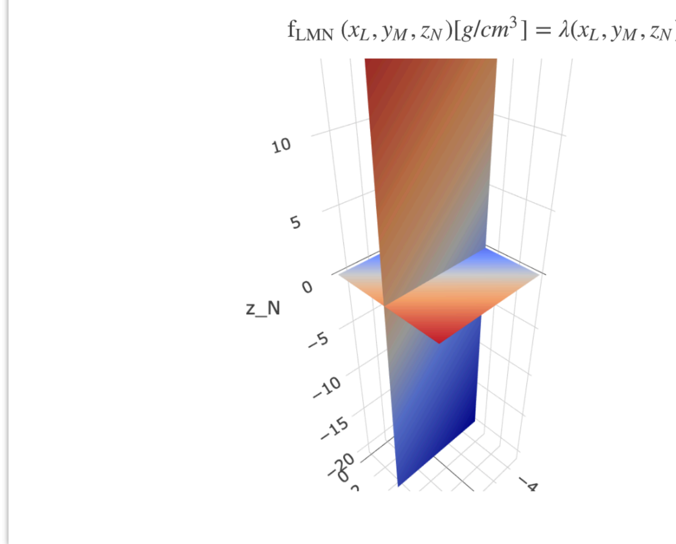
Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

nargs	arg shapes	out shape	Plot Type	notes
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Map-to-plane

```
iplot(kamodo.plot(f_LMN = dict(z_N = 0),
                  g_LMN = dict(y_M = 5)))
```



Visualization API

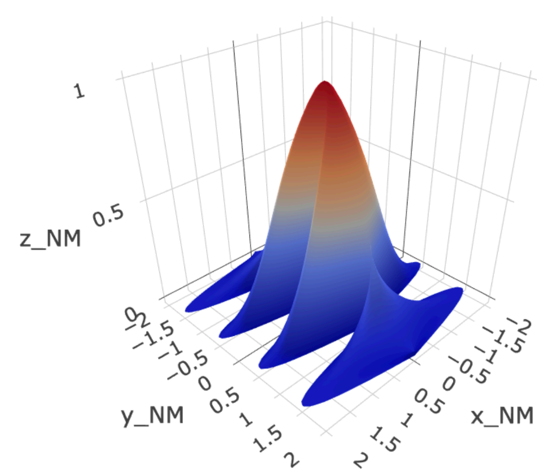
Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Parametric Surface

```
iplot(kamodo.plot(p = {}), show_link = False)
```

$$p(x_{NM}, y_{NM}, z_{NM})[cm] = \lambda(x_{NM}, y_{NM}, z_{NM})$$

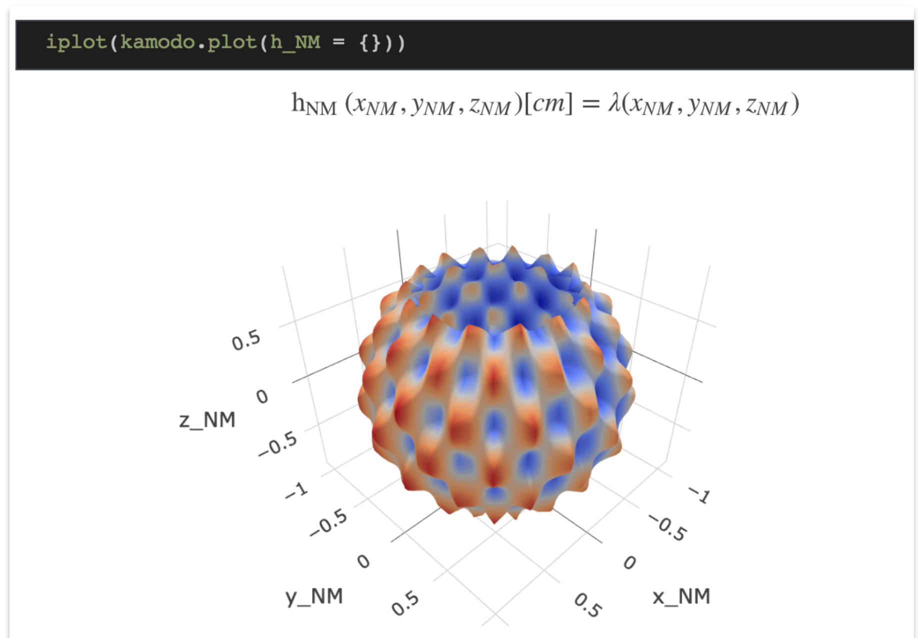


Visualization API

Kamodo generates assets ready for visualization based on the array shapes of field inputs and outputs. Kamodo currently outputs Plotly (json) for interactivity and easy web deployment.

			Plot Type	notes
nargs	arg shapes	out shape		
1	N	N	1-d line	
		Nx2	2-d line	
		Nx3	3-d line	
	Nx2	Nx2	2-d vector field	
	Nx3	Nx3	3-d vector field	
2	N, M	NxM	2-d contour	indexing
	NxM, NxM	NxM	2-d contour (skew/carpet)	indexing
3	1, M, N	MxN	Map-to-plane	indexing*
	L, 1, N	LxN	Map-to-plane	indexing*
	L, M, 1	LxM	Map-to-plane	indexing*
	N, N, N	N	3-d colored line	
	NxM, NxM, NxM	1	Parametric Surface	
		NxM	Coloured Parametric Surface	

Colored Parametric Surface



Kamodofication

- the process of exposing models and data to Kamodo

A model or data source is considered “kamodofied” when all scientifically relevant variables are exposed as Kamodo objects.

Kamodofication requirements:

1. Model must be accessible from python
2. Model must provide an interpolating function for each variable
3. Interpolating functions should supply default values as arguments, indicating the valid domain for their inputs.
4. Variable names should follow Kamodo’s naming specification for LaTeX legibility.
5. Interpolating functions must contain the following metadata as attributes:
    1. meta - dictionary of {‘units’: ’kg’, ‘citation’ : ’Doe, J. et. al’}
    2. data - array
6. Class Methods should use “self” as the first argument.

Kamodofied Models

TIEGCM

Thermosphere Ionosphere Electrodynamics General Circulation Model

R. G. Roble et al., High Altitude Observatory, National Center for Atmospheric Research

$\mathrm{TN}(time, ilev, lat, lon)[K] = \lambda(time, ilev, lat, lon)$
$\mathrm{UN}(time, ilev, lat, lon)[cm/s] = \lambda(time, ilev, lat, lon)$
$\mathrm{VN}(time, ilev, lat, lon)[cm/s] = \lambda(time, ilev, lat, lon)$
$\mathrm{O_1}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{NO}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{N4S}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{HE}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{NE}(time, ilev, lat, lon)[cm**-3] = \lambda(time, ilev, lat, lon)$
$\mathrm{TE}(time, ilev, lat, lon)[K] = \lambda(time, ilev, lat, lon)$
$\mathrm{TI}(time, ilev, lat, lon)[K] = \lambda(time, ilev, lat, lon)$
$\mathrm{O_2}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{O2P_{ELD}}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\omega(time, ilev, lat, lon)[s**-1] = \lambda(time, ilev, lat, lon)$
$\mathrm{POTEN}(time, ilev, lat, lon)[volts] = \lambda(time, ilev, lat, lon)$
$\mathrm{UI_{ExB}}(time, ilev, lat, lon)[cm/s] = \lambda(time, ilev, lat, lon)$
$\mathrm{VI_{ExB}}(time, ilev, lat, lon)[cm/s] = \lambda(time, ilev, lat, lon)$
$\mathrm{WI_{ExB}}(time, ilev, lat, lon)[cm/s] = \lambda(time, ilev, lat, lon)$
$\mathrm{OP}(time, ilev, lat, lon)[cm**-3] = \lambda(time, ilev, lat, lon)$
$\mathrm{N2P_{ELD}}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\mathrm{NPLUS}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\mathrm{NOP_{ELD}}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\sigma_{PED}(time, ilev, lat, lon)[ohm**-1/m] = \lambda(time, ilev, lat, lon)$
$\sigma_{HAL}(time, ilev, lat, lon)[ohm**-1/m] = \lambda(time, ilev, lat, lon)$
$\mathrm{DEN}(time, ilev, lat, lon)[g/cm**3] = \lambda(time, ilev, lat, lon)$
$\mathrm{QJOULE}(time, ilev, lat, lon)[erg/(g*s)] = \lambda(time, ilev, lat, lon)$
$Z(ilev, time, lat, lon, z_{level})[cm] = \lambda(ilev, time, lat, lon, z_{level})$
$\mathrm{ZG}(time, ilev, lat, lon)[cm] = \lambda(time, ilev, lat, lon)$
$\mathrm{O_{N2}}(time, ilev, lat, lon)[1] = \lambda(time, ilev, lat, lon)$
$\mathrm{N2D_{ELD}}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\mathrm{O2N}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\mathrm{N2N}(time, ilev, lat, lon) = \lambda(time, ilev, lat, lon)$
$\mathrm{TEC}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{QJOULE_{INTEG}}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{EFLUX}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{HMF_2}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{NMF_2}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{TLBC}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{ULBC}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{VLBC}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{TLBC_{NM}}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{ULBC_{NM}}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{VLBC_{NM}}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{latitude}(time, lat, lon) = \lambda(time, lat, lon)$
$\mathrm{longitude}(time, lat, lon) = \lambda(time, lat, lon)$

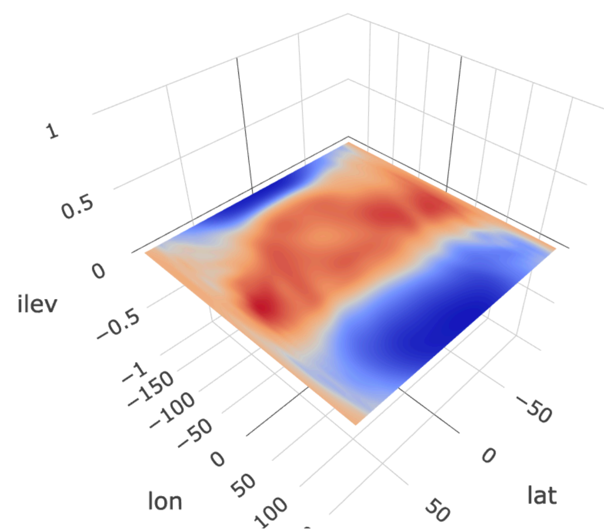
Kamodofied TIEGCM fields. Interpolator functions provided by scipy. Dimension reduction is achieved by specifying a value for time.

```
kamodo['DEN_250(lat, lon, ilev)'] = 'DEN(250, ilev, lat, lon)'
```

$$\mathrm{DEN}_{250}(lat, lon, ilev) = \mathrm{DEN}(250, ilev, lat, lon)$$

```
llat, llon = np.meshgrid(lat, lon)
iplot(kamodo.plot(DEN_250 = dict(lat = llat, lon = llon, ilev = 0)))
```

$$\mathrm{DEN}_{250}(lat, lon, ilev) = \mathrm{DEN}(250, ilev, lat, lon)$$



Kamodofied Models

GITM

Global Ionosphere Thermosphere Model

A.J. Ridley, Department of Atmosphere, Oceanic and Space Sciences, University of Michigan

Access to GITM is provided by SpacePy, interpolation methods built on SciPy.

$\rho(lon, lat, alt)[kg/m^3] = \lambda(lon, lat, alt)$

$\mathrm{Altitude}\,(lon, lat, alt) = \lambda(lon, lat, alt)$

$\mathrm{NeutralTemperature}\,(lon, lat, alt)[K] = \lambda(lon, lat, al$

$\mathrm{V}_{\mathrm{n}}^{\mathrm{up}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

$\mathrm{Latitude}\,(lon, lat, alt) = \lambda(lon, lat, alt)$

$\mathrm{V}_{\mathrm{i}}^{\mathrm{east}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

$\mathrm{NO}\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{N}_2\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{NO}^+\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{O\,(3P)}\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{lt}\,(lon, lat, alt)[hours] = \lambda(lon, lat, alt)$

$\mathrm{O}_{\mathrm{4SP}}^+\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{dLon}\,(lon, lat, alt) = \lambda(lon, lat, alt)$

$\mathrm{e\text{-}}\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{Longitude}\,(lon, lat, alt) = \lambda(lon, lat, alt)$

$\mathrm{V}_{\mathrm{i}}^{\mathrm{up}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

$\mathrm{N}_2^+\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{V}_{\mathrm{i}}^{\mathrm{north}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

$\mathrm{V}_{\mathrm{n}}^{\mathrm{east}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

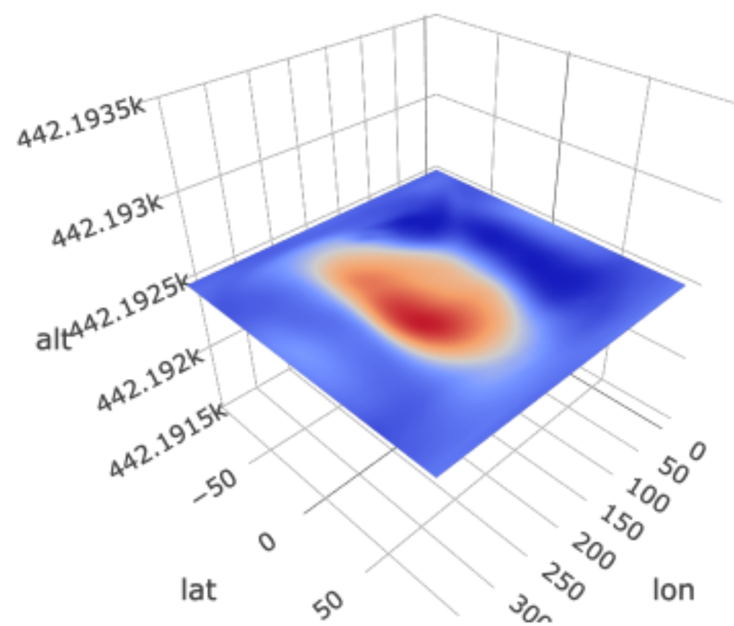
$\mathrm{O}_2^+\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{dLat}\,(lon, lat, alt) = \lambda(lon, lat, alt)$

$\mathrm{O}_2\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$

$\mathrm{V}_{\mathrm{n}}^{\mathrm{north}}\,(lon, lat, alt)[m/s] = \lambda(lon, lat, alt)$

$\mathrm{O\,(3P)}\,(lon, lat, alt)[1/m^3] = \lambda(lon, lat, alt)$



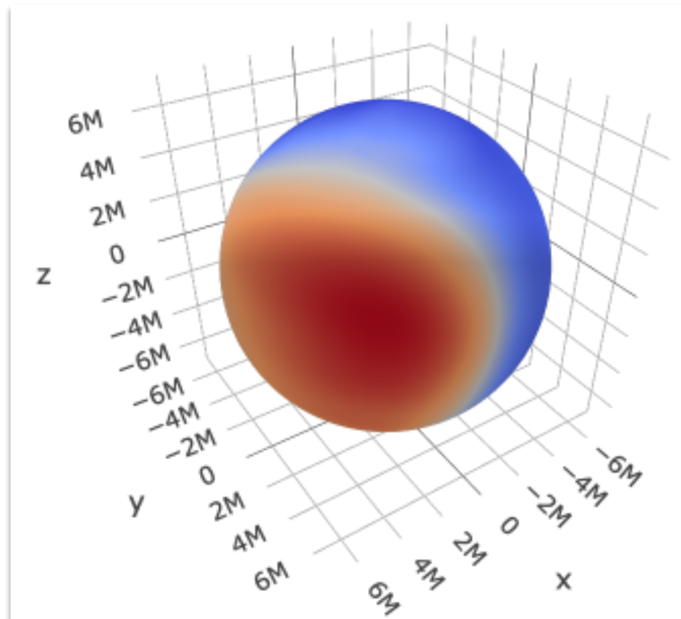
Kamodofied Models

GITM

Global Ionosphere Thermosphere Model

A.J. Ridley, Department of Atmosphere, Oceanic and Space Sciences, University of Michigan

Transformation to cartesian coordinates is achieved through function composition.



$$\text{ALT}(r)[m] = \lambda(r)$$
$$\text{LON}(\phi)[degrees] = \lambda(\phi)$$
$$\text{LAT}(\theta)[degrees] = \lambda(\theta)$$
$$r(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$
$$\theta(x, y, z) = \operatorname{asin}\left(\frac{z}{r(x, y, z)}\right)$$
$$\phi(x, y) = \operatorname{atan}_2(y, x)$$

$$\rho(x, y, z)[kilogram/meter**3] = \rho(\text{LON}(\phi(x, y)), \text{LAT}(\theta(x, y, z)), \text{ALT}(r(x, y, z)))$$
$$\text{O(3P)}(x, y, z)[meter**(-3)] = \text{O(3P)}(\text{LON}(\phi(x, y)), \text{LAT}(\theta(x, y, z)), \text{ALT}(r(x, y, z)))$$

Kamodofied Models

Adaptively Refined MHD Solver (ARMS)

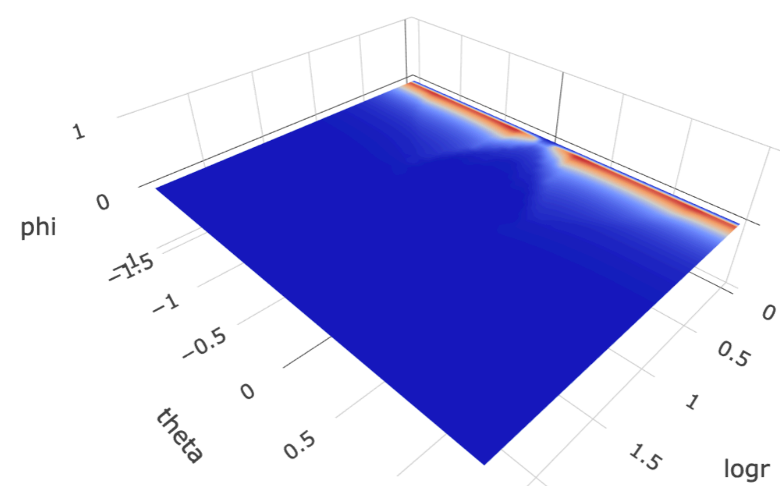
Antiochos, S. K., Masson, S. DeVore, Goddard GSFC

ARMS is a 3D solar physics model capable of capturing transient solar eruptions. We have kamodofied the following fields from ARMS, using a custom octree interpolator written in python. Kamodo is used to transform from the model coordinates to cartesian.

$$\rho(logr, \theta, \phi)[g/cm^3] = \lambda(logr, \theta, \phi)$$
$$\mathrm{v_r}(logr, \theta, \phi)[cm/s] = \lambda(logr, \theta, \phi)$$
$$\mathrm{v}_\theta(logr, \theta, \phi)[cm/s] = \lambda(logr, \theta, \phi)$$
$$\mathrm{v}_\phi(logr, \theta, \phi)[cm/s] = \lambda(logr, \theta, \phi)$$
$$P(logr, \theta, \phi)[Pa] = \lambda(logr, \theta, \phi)$$
$$T(logr, \theta, \phi)[K] = \lambda(logr, \theta, \phi)$$
$$\mathrm{B_r}(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\mathrm{B}_\theta(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\mathrm{B}_\phi(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\mathrm{J_r}(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\mathrm{J}_\theta(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\mathrm{J}_\phi(logr, \theta, \phi)[G] = \lambda(logr, \theta, \phi)$$
$$\beta(logr, \theta, \phi)[1] = \lambda(logr, \theta, \phi)$$
$$\mathrm{s_{alfven}}(logr, \theta, \phi)[1] = \lambda(logr, \theta, \phi)$$
$$\Phi(logr, \theta, \phi)[1] = \lambda(logr, \theta, \phi)$$

```
iplot(arms.plot(rho = dict( logr = llogr, theta = ttheta, phi = 0)))
```

$$\rho(logr, \theta, \phi)[g/cm^3] = \lambda(logr, \theta, \phi)$$



Kamodofied Models

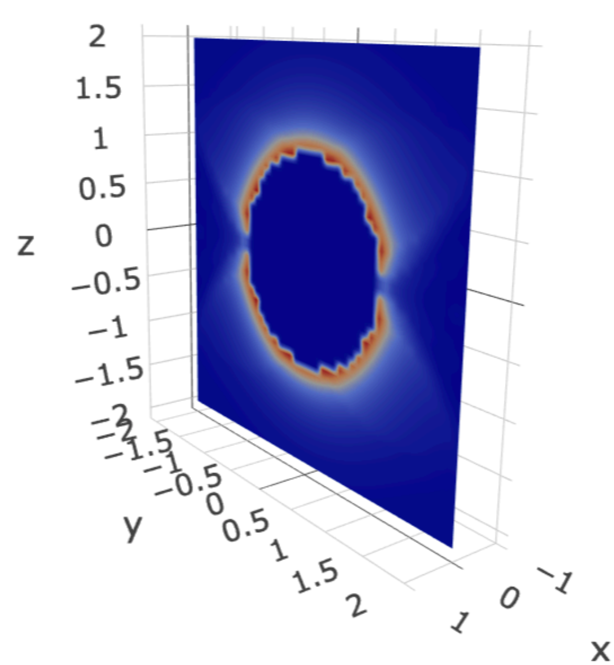
Adaptively Refined MHD Solver (ARMS)

Antiochos, S. K., Masson, S. DeVore, Goddard GSFC

ARMS is a 3D solar physics model capable of capturing transient solar eruptions. We have kamodofied the following fields from ARMS, using a custom octree interpolator written in python. Kamodo is used to transform from the model coordinates to cartesian.

```
iplot(arms.plot(RHO = dict( x = 0, y = yy, z = zz)))
```

$$r(x, y, z) = \sqrt{x^2 + y^2 + z^2}$$
$$\operatorname{logr}(x, y, z)[log(Dimension(1))] = \log(r(x, y, z))$$
$$\theta(x, y, z) = \operatorname{asin}\left(\frac{z}{r(x, y, z)}\right)$$
$$\phi(x, y) = \operatorname{atan}_2(y, x)$$
$$\rho(x, y, z)[gram/centimeter**3] = \rho(\operatorname{logr}(x, y, z), \theta(x, y, z), \phi(x, y))$$



Trick: rho and RHO are unique python names that render as the same greek letter in LaTeX, so we are free to define both rho(x,y,z) and RHO(logr,theta,phi), different signatures for the same variable.

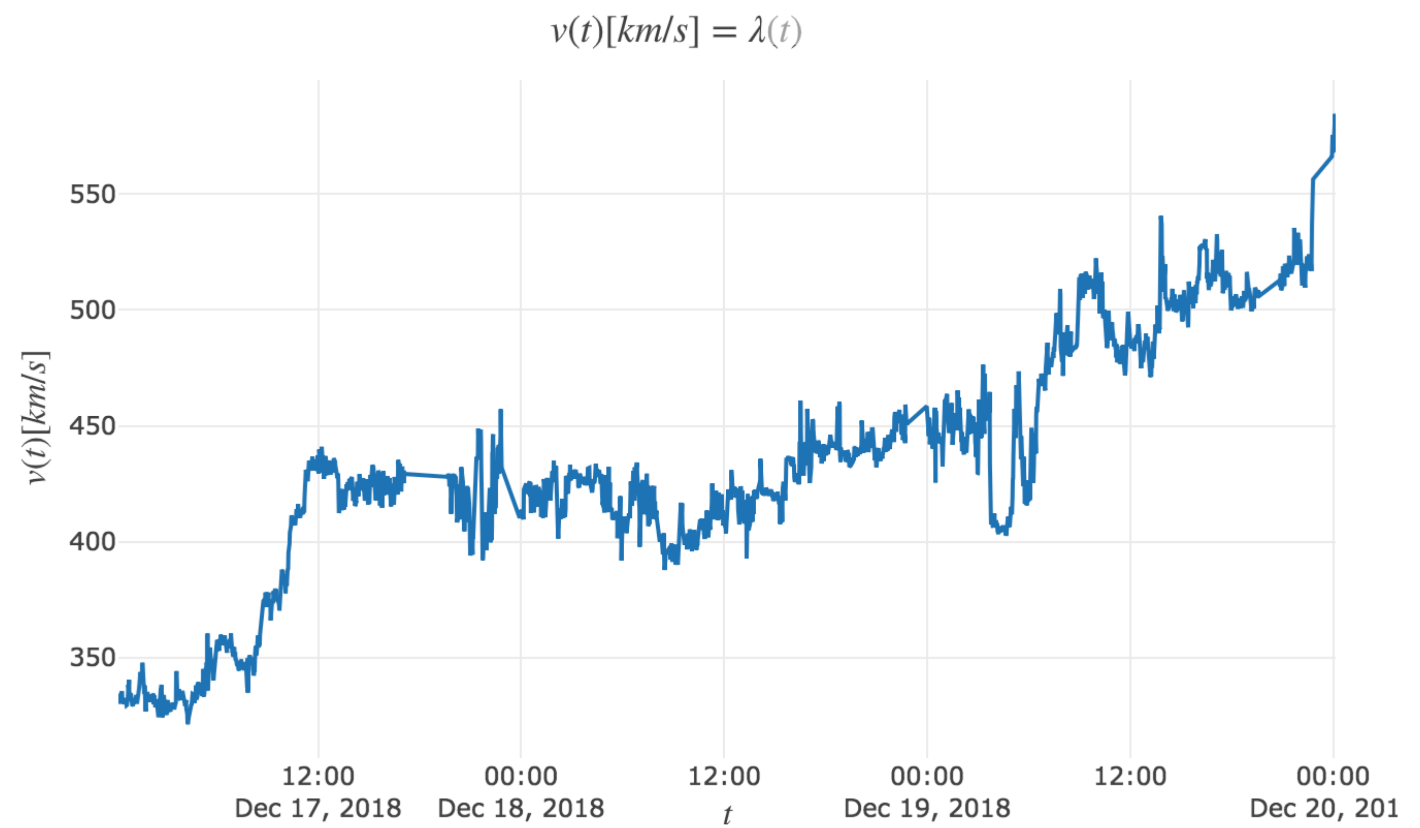
Kamodofied Data: DISCOVR

The Kamodofied discovr feed provides interpolated plasma and field variables. The time interpolator is built on pandas' time series interpolators.

```
dscovr_url = 'http://services.swpc.noaa.gov/products/solar-wind/'
dscovr = DSCOVR(dscovr_url, verbose = False)
dscovr
```

$\text{status}(t) = \lambda(t)$
$v(t)[km/s] = \lambda(t)$
$T(t)[K] = \lambda(t)$
$\rho(t)[1/cm^3] = \lambda(t)$
$b(t)[nT] = \lambda(t)$
$\text{lat}_{\text{gsm}}(t)[degrees] = \lambda(t)$
$\text{bz}_{\text{gsm}}(t)[nT] = \lambda(t)$
$\text{lon}_{\text{gsm}}(t)[degrees] = \lambda(t)$
$\text{by}_{\text{gsm}}(t)[nT] = \lambda(t)$
$\text{bx}_{\text{gsm}}(t)[nT] = \lambda(t)$

```
iplot(dscovr.figure('v'), show_link = False)
```



Kamodofied Data: HAPI

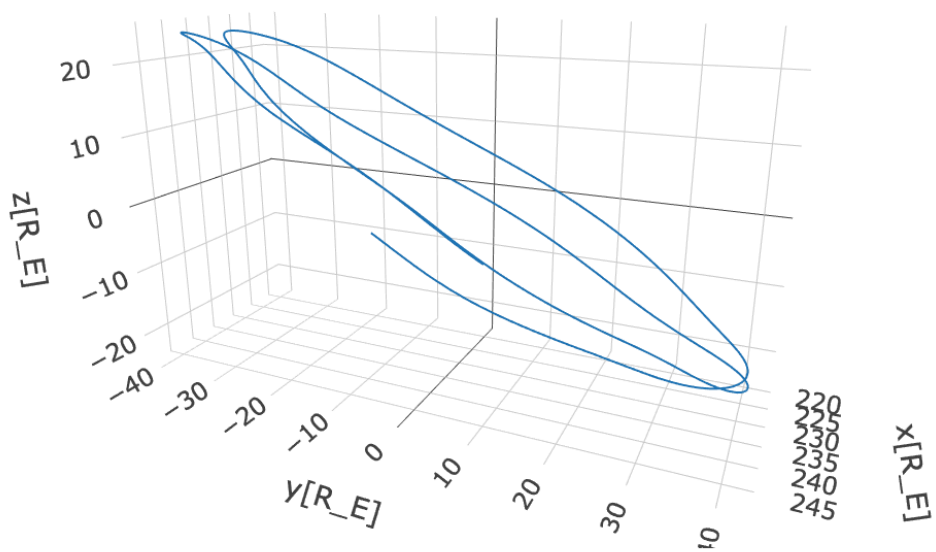
A work-in-progress, the Kamodofied HAPI api is built on the python hapiclient from Bob Weigel. Here, the variable *xvec* is built from the position components of ACE spacecraft as a function of time.

```
kamodo
```

$\mathrm{X_{GSE}}(t)[R_E] = \lambda(t)$
$\mathrm{Y_{GSE}}(t)[R_E] = \lambda(t)$
$\mathrm{Z_{GSE}}(t)[R_E] = \lambda(t)$
$\vec{x}_{GSE}(t)[R_E] = \lambda(t)$

```
iplot(kamodo.figure('xvec_GSE', t = t), show_link = False)
```

$$\vec{x}_{GSE}(t)[R_E] = \lambda(t)$$



Summary/Future Plans

tldr: Kamodo provides a functional, publication-focused interface for space weather models and data.

Further Resources:

Project Page - https://ccmc.gsfc.nasa.gov/Kamodo/

Code - https://sed-gitlab.gsfc.nasa.gov/ccmc/Kamodo

Future:

Output formats - An extension of visualization API (csv, OpenSpace, json)

Komodo-Live - Automatically generate web-based interfaces similar to Kameleon-Live

Packaging - Provide conda environments for easy distribution

OpenSource - NASA Software Release Process is underway

Thank You!

Kamodo Functional API

Kamodo will interpret certain strings as LaTeX for “syntactic sugar”:

```
kamodo = Kamodo(
        'rho = ALPHA+BETA+GAMMA',
        'rvec = t',
        'fprime = x',
        'xvec_i = xvec_iminus1 + 1',
        'F__gravity = G*M*m/R**2',
  )
```

$$\rho(\alpha,\beta,\gamma) = \alpha + \beta + \gamma$$
$$\vec{r}(t) = t$$
$$\operatorname{f'}(x) = x$$
$$\vec{x}_i\left(\vec{x}_{i-1}\right) = \vec{x}_{i-1} + 1$$
$$\operatorname{F^{gravity}}(G,M,R,m) = \frac{GMm}{R^2}$$

Corresponding LaTeX output is available for use in publications:

```
kamodo.to_latex()
```

```
u"\\begin{equation}\\rho{\\left (\\alpha,\\beta,\\gamma \\right )} = \\alpha + \\beta + \\gamma\\end{equation}\\begin{equation}\\vec{r}{\\left (t \\right )} = t\\end{equation}\\begin{equation}\\operatorname{{f}'}{\\left (x \\right )} = x\\end{equation}\\begin{equation}\\vec{x}_{i}{\\left (\\vec{x}_{i-1} \\right )} = \\vec{x}_{i-1} + 1\\end{equation}\\begin{equation}\\operatorname{F^{gravity}}{\\left (G,M,R,m \\right )} = \\frac{G M m}{R^{2}}\\end{equation}"
```

Fortran -> Kamodo

Models must be accessible as python modules to be compatible with Kamodo. Here, a Fortran reader is used to subclass Kamodo.

Developer

Python Kamodo-compatible class

```python
import read_ascii
from scipy.interpolate import RegularGridInterpolator
import numpy as np

class ColumnReader(Kamodo):
    def __init__(self, filename, *args, **kwargs):
        super(ColumnReader, self).__init__(*args, **kwargs)

        read_ascii.ascii.read_file(filename)
        self.lon = read_ascii.ascii.lons
        self.lat = read_ascii.ascii.lats
        self.alt = read_ascii.ascii.alts
        self.data = read_ascii.ascii.data

        bounds_error = kwargs.get('bounds_error', False)
        fill_value = kwargs.get('missing_value', np.nan)
        self.interpolator = RegularGridInterpolator((self.alt, self.lat, self.lon),
                                                    self.data, bounds_error = False)
        try:
            self['rho'] = self.density
        except:
            print self.signatures
            raise

    @kamodofy(units = '1/cm^3')
    def density(self, alt, lat, lon):
        points = np.hstack((alt, lat, lon)).reshape(3, -1).T
        return self.interpolator(points)
```

Fortran Reader

```fortran
! FILE: read_ascii.f90
module ascii
  INTEGER :: nlon,nlat,nalt
  REAL*8, allocatable, dimension(:) ::
ALTS,LATS,LONS
  REAL*8, allocatable, dimension(:,:,:) ::
data
contains
   SUBROUTINE read_file(FILENAME)
!
!     READ ASCII FILE with interpolated data
from TIE-GCM
!
```

Fortran readers and interpolators may be accessed through the popular F2Py utility

User

```python
filename = 'sample_data/SampleFortran/Sophia_Schwalbe_082718_IT_1__TIE-GCM__20120709_002000.dat'
column = ColumnReader(filename)
column
```

$\rho(alt, lat, lon)[1/cm^3] = \lambda(alt, lat, lon)$

```python
column.rho(331, 33.5, -106.35)
```

```
array([ 2526809.55244755])
```

Custom readers inherit all the functionality of Kamodo

Kamodo Analysis

If numerical solutions are not available, as in the case of field line integration, Kamodo can use scipy to solve initial value problems, which are a class of ordinary differential equations.

```
decay = Kamodo(fprime = lambda t, x: -.9*x)

@event
def stop_condition(t,x):
    return x[0] - .5

decay['f'] = decay.solve('fprime', # rhs ODE
                         interval = [0, 1], #integration limits
                         y0 = [1], #initial  condition
                         events = stop_condition,
                        )
decay
```

$\mathrm{f}'(t,x) = \lambda(t,x)$

$f(t) = \lambda(t)$

A stop condition may be triggered by returning a negative value

Kamodo Analysis

If numerical solutions are not available, as in the case of field line integration, Kamodo can use scipy to solve initial value problems, which are a class of ordinary differential equations.

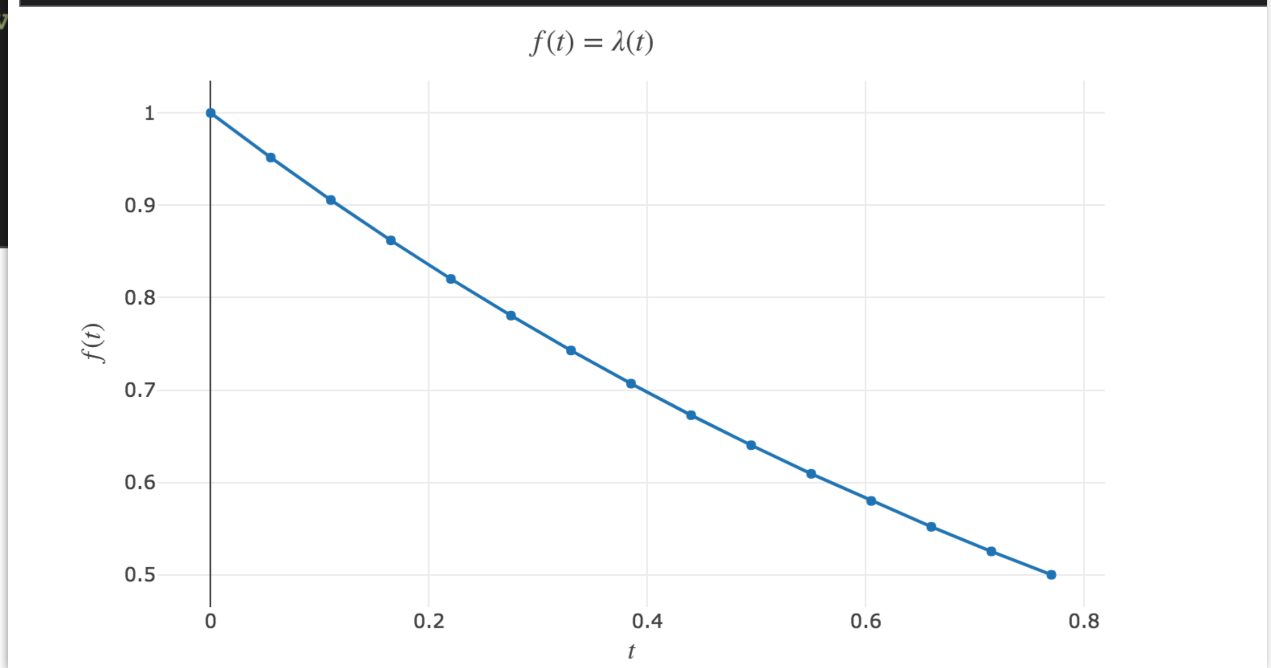
```
decay = Kamodo(fprime = lambda t, x: -.9*x)

@event
def stop_condition(t,x):
    return x[0] - .5

decay['f'] = decay.solv



decay
```

A stop condition may be triggered by returning a negative value

$f'(t,x) = \lambda(t,x)$

$f(t) = \lambda(t)$

```
iplot(decay.plot(f = dict(t = np.linspace(tinitial, tfinal, 15))), show_link = False)
```



Kamodo Analysis

For vector-valued problems, treat each component separately. This framework will allow us to generate integral curves, including fieldlines, streaklines, particle trajectories, etc.

```
@event
def stop(t,y):
    return .35 - y[2]

def fvecprime(t, yvec):
    return [np.sin(np.pi*t), np.cos(np.pi*t), .1]

test = Kamodo(fvecprime = fvecprime)
test['fvec'] = test.solve('fvecprime',
                          [0,5], # interval
                          np.array([0, 0, 0]), # initial condition
                          events = event(stop)) # terminating event
test
```

Stop when z = .35

$$\vec{f}'(t, \vec{y}) = \lambda(t, \vec{y})$$

$$\vec{f}(t) = \lambda(t)$$

Kamodo Analysis

For vector-valued problems, treat each component separately. This framework will allow us to generate integral curves, including fieldlines, streaklines, particle trajectories, etc.

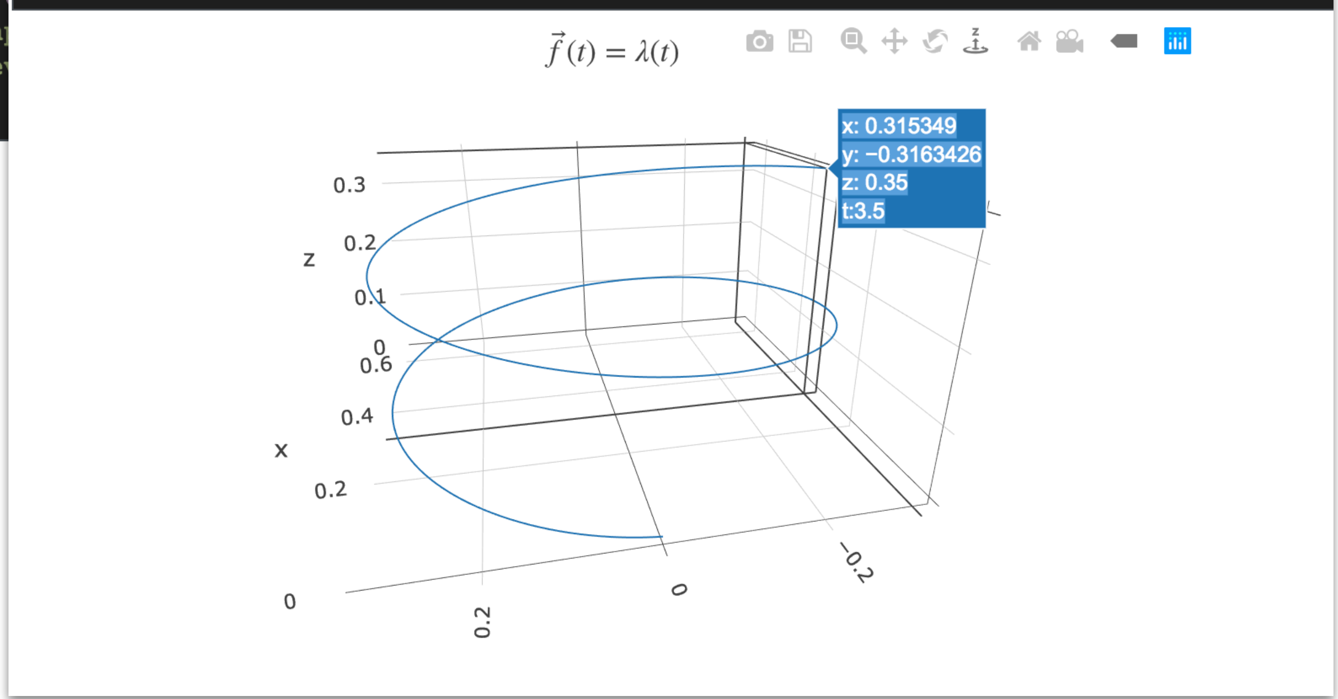
```
@event
def stop(t,y):
    return .35 - y[2]

def fvecprime(t, yvec):
    return [np.sin(np.pi*t), np.cos(np.pi*t), .1]

test = Kamodo(fvecprime = fvecprime)
test['fvec'] = test.solv
                         [(
                         np
                         ev
test
```

Stop when z = .35

$\vec{f}'(t,\vec{y}) = \lambda(t,\vec{y})$

$\vec{f}(t) = \lambda(t)$

```
iplot(test.plot(fvec = dict(t = np.linspace(tinitial,tfinal, 303))), show_link = False)
```



Next Steps with OpenSpace

Now that we can quickly expose space weather models and data to scientists and educators, we can leverage this in the OpenSpace astrovisualization engine.

We need a target format for assets optimized in OpenSpace. Either

- have Kamodo output to that format
- convert from Plotly's json
- lupa (python wrapper for Lua/LuaJit)

We would like a discussion of pros and cons of either approach.