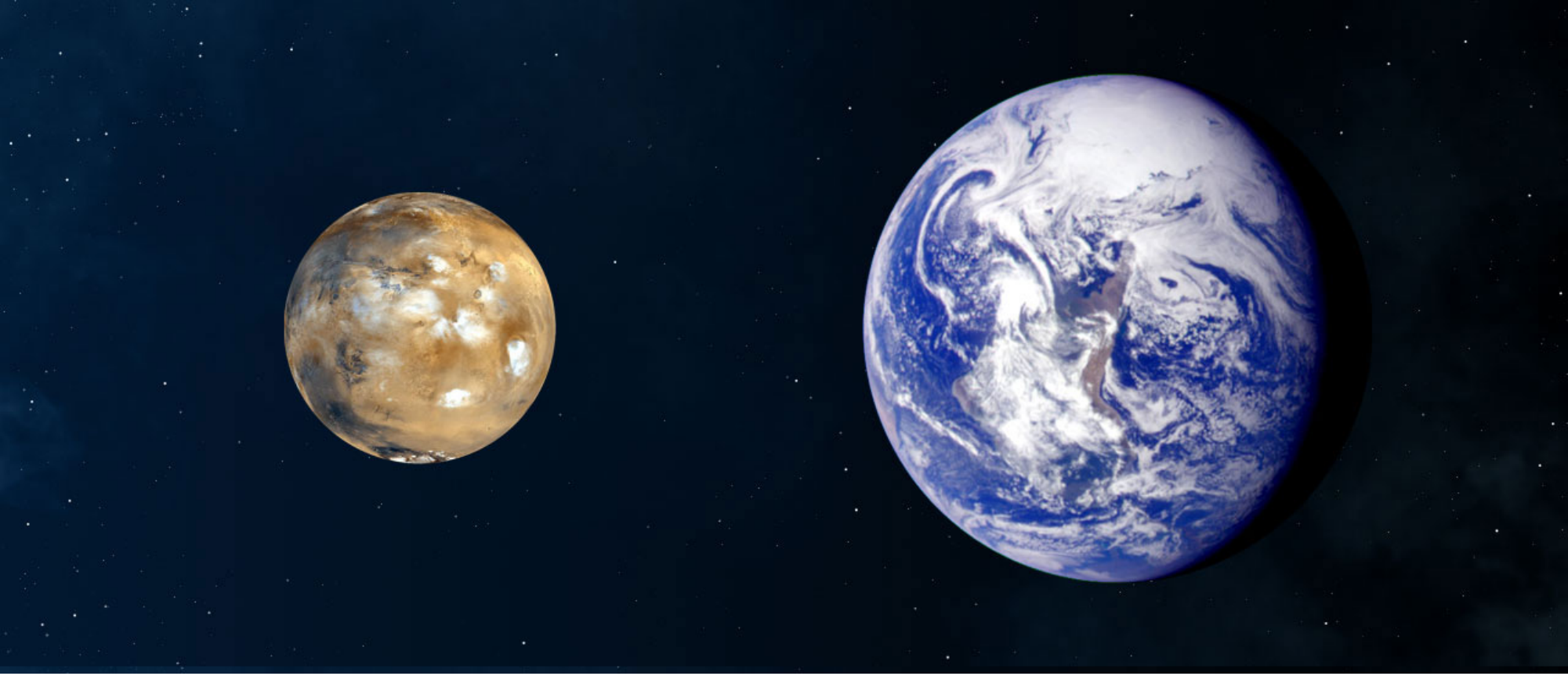


Py-Earth: Multivariate Adaptive Regression Splines (*MARS*) in Python

Mehdi Cherti (Appstat, LAL/CNRS)
Supervised by Balázs Kégl (LAL/CNRS) and
Alexandre Gramfort (CNRS LTCI)

October 26, 2015

Copyright © Walter Myers

Introduction

- **MARS** is a regression technique for high dimensional data:
  - introduced first by Jerome H.Friedman in 1991
- it is non-linear and non-parametric
- **Py-earth** is an implementation of **MARS** in Python, created by **Jason Crudy**
- The goal is to bring **Py-earth** into scikit-learn

Copyright © Walter Myers

Introduction

- **MARS** is a regression technique for high dimensional data:
  - introduced first by Jerome H.Friedman in 1991
- it is non-linear and non-parametric
- **Py-earth** is an implementation of **MARS** in Python, created by **Jason Crudy**
- The goal is to bring **Py-earth** into scikit-learn

Copyright © Walter Myers

Introduction

- **MARS** is a regression technique for high dimensional data:
  - introduced first by Jerome H.Friedman in 1991
- it is non-linear and non-parametric
- **Py-earth** is an implementation of **MARS** in Python, created by Jason Crudy
- The goal is to bring Py-earth into scikit-learn

Copyright © Walter Myers

Introduction

- **MARS** is a regression technique for high dimensional data:
  - introduced first by Jerome H.Friedman in 1991
- it is non-linear and non-parametric
- **Py-earth** is an implementation of **MARS** in Python, created by **Jason Crudy**
- The goal is to bring Py-earth into scikit-learn

Copyright © Walter Myers

Introduction

- **MARS** is a regression technique for high dimensional data:
  - introduced first by Jerome H.Friedman in 1991
- it is non-linear and non-parametric
- **Py-earth** is an implementation of **MARS** in Python, created by **Jason Crudy**
- The goal is to bring **Py-earth** into scikit-learn

Copyright © Walter Myers

Setup

- **Setup** : Multivariate regression with multiple outputs : (X_i, Y_i) where
  - **i** is the ith example
  - X_i is a **vector** describing each example **i**
  - Y_i is a **real-valued vector** describing the true outputs of the example **i**
- We want to find a model f which predicts **Y** from **X** with low generalization mean squared error (**MSE**)

Copyright © Walter Myers

Setup

- **Setup** : Multivariate regression with multiple outputs : (X_i, Y_i) where
  - **i** is the ith example
  - X_i is a **vector** describing each example **i**
  - Y_i is a **real-valued vector** describing the true outputs of the example **i**
- We want to find a model f which predicts **Y** from **X** with low generalization mean squared error (**MSE**)

Copyright © Walter Myers

Setup

- **Setup** : Multivariate regression with multiple outputs : (X_i, Y_i) where
  - **i** is the ith example
  - X_i is a **vector** describing each example **i**
  - Y_i is a **real-valued vector** describing the true outputs of the example **i**
- We want to find a model f which predicts **Y** from **X** with low generalization mean squared error (**MSE**)

Copyright © Walter Myers

Setup

- **Setup** : Multivariate regression with multiple outputs : (X_i, Y_i) where
  - **i** is the ith example
  - X_i is a **vector** describing each example **i**
  - Y_i is a **real-valued vector** describing the true outputs of the example **i**
- We want to find a model f which predicts **Y** from **X** with low generalization mean squared error (**MSE**)

Copyright © Walter Myers

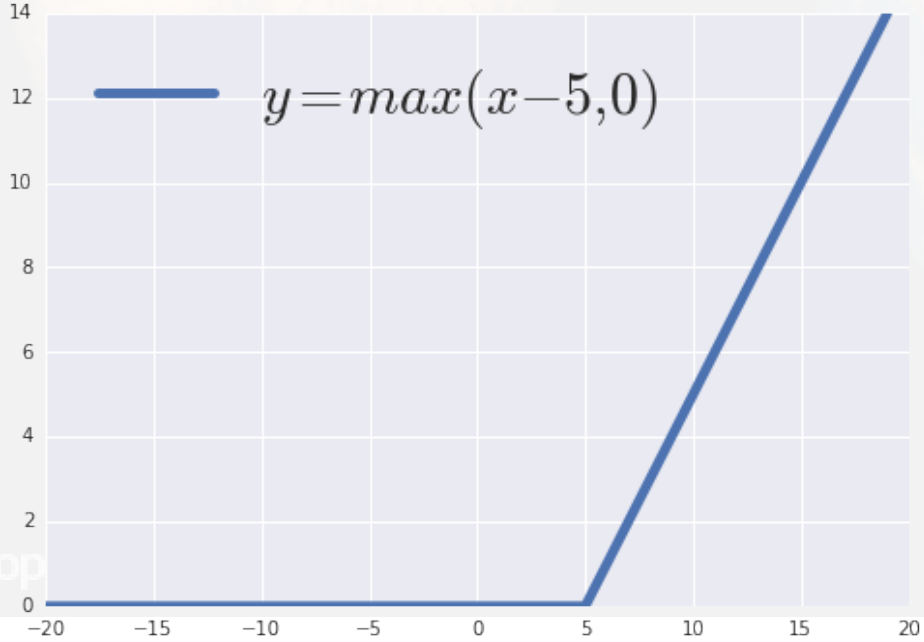
How does MARS work ?

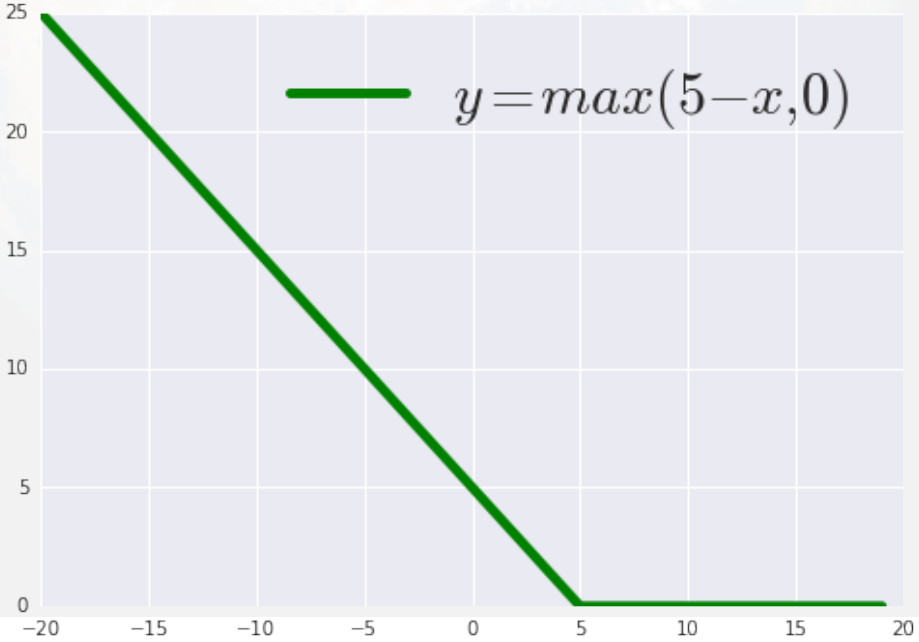
Basic building block : Hinge functions,

$$y = max(x - k, 0)$$

or

$$y = max(k - x, 0)$$





How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- The algorithm constructs adaptively a set of basis functions : $B_k(X)$
- Each basis function is a product of hinge functions, for instance :
  - $B_k(X) = max(X_1 - 5, 0)max(X_2 + 4, 0)$
- The model is a linear combination of those basis functions
  - $Y = \sum_{k=1}^{K} \alpha_k B_k(X)$
- The specificity of **MARS** comes from how the basis functions are created and added to the model

Copyright © Walter Myers

How does MARS work ?

- Two steps, the forward pass and the pruning pass
- We over-generate a set of basis functions in the forward pass
- We prune unncessary basis functions in the pruning pass

Copyright © Walter Myers

How does MARS work ?

- Two steps, the forward pass and the pruning pass
- We over-generate a set of basis functions in the forward pass
- We prune unncessary basis functions in the pruning pass

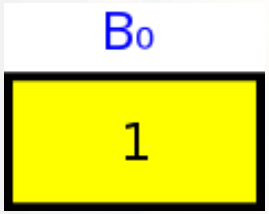
Copyright © Walter Myers

How does MARS work ?

- Two steps, the forward pass and the pruning pass
- We over-generate a set of basis functions in the forward pass
- We prune unncessary basis functions in the pruning pass

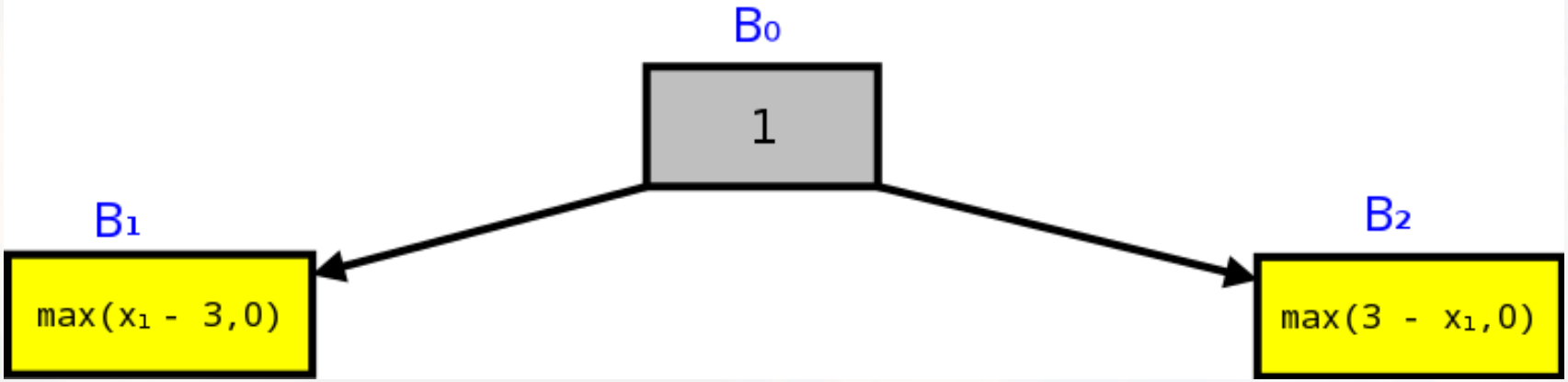
Copyright © Walter Myers

How does MARS work ? The forward pass



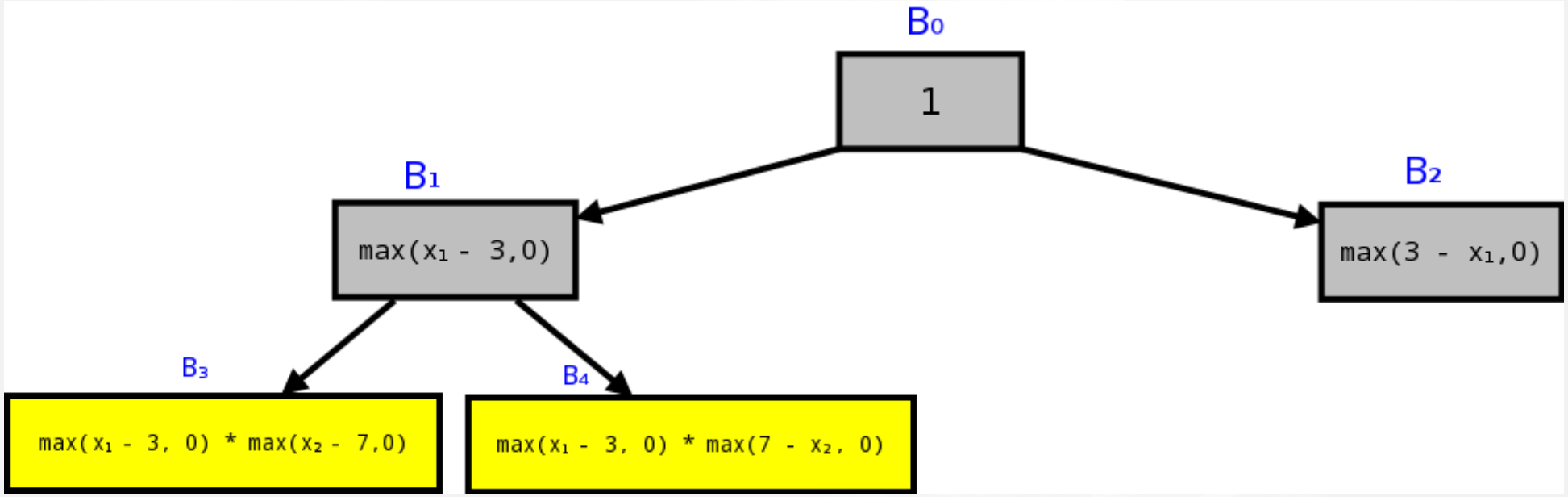
Copyright © Walter Myers

How does MARS work ? The forward pass



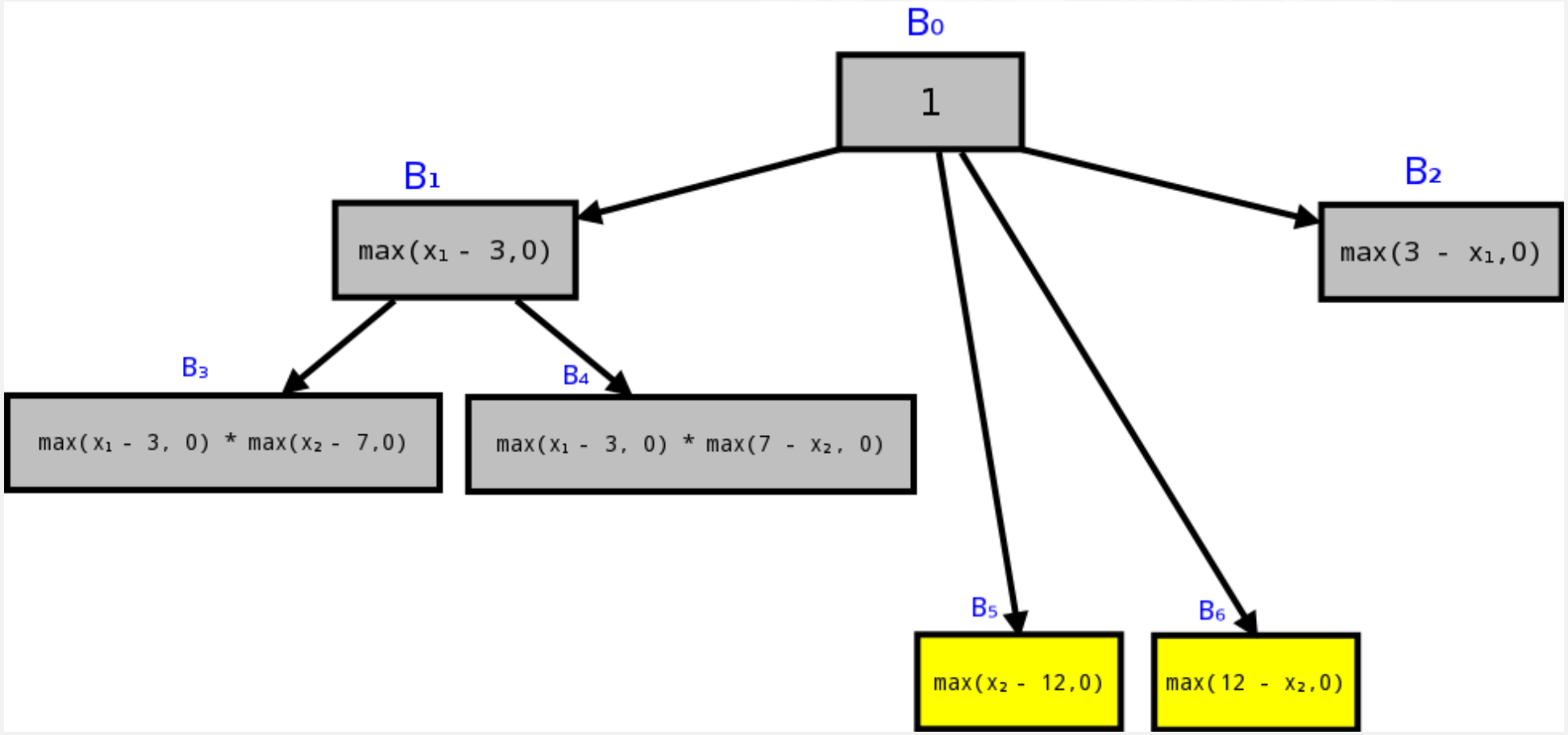
Copyright © Walter Myers

How does MARS work ? The forward pass



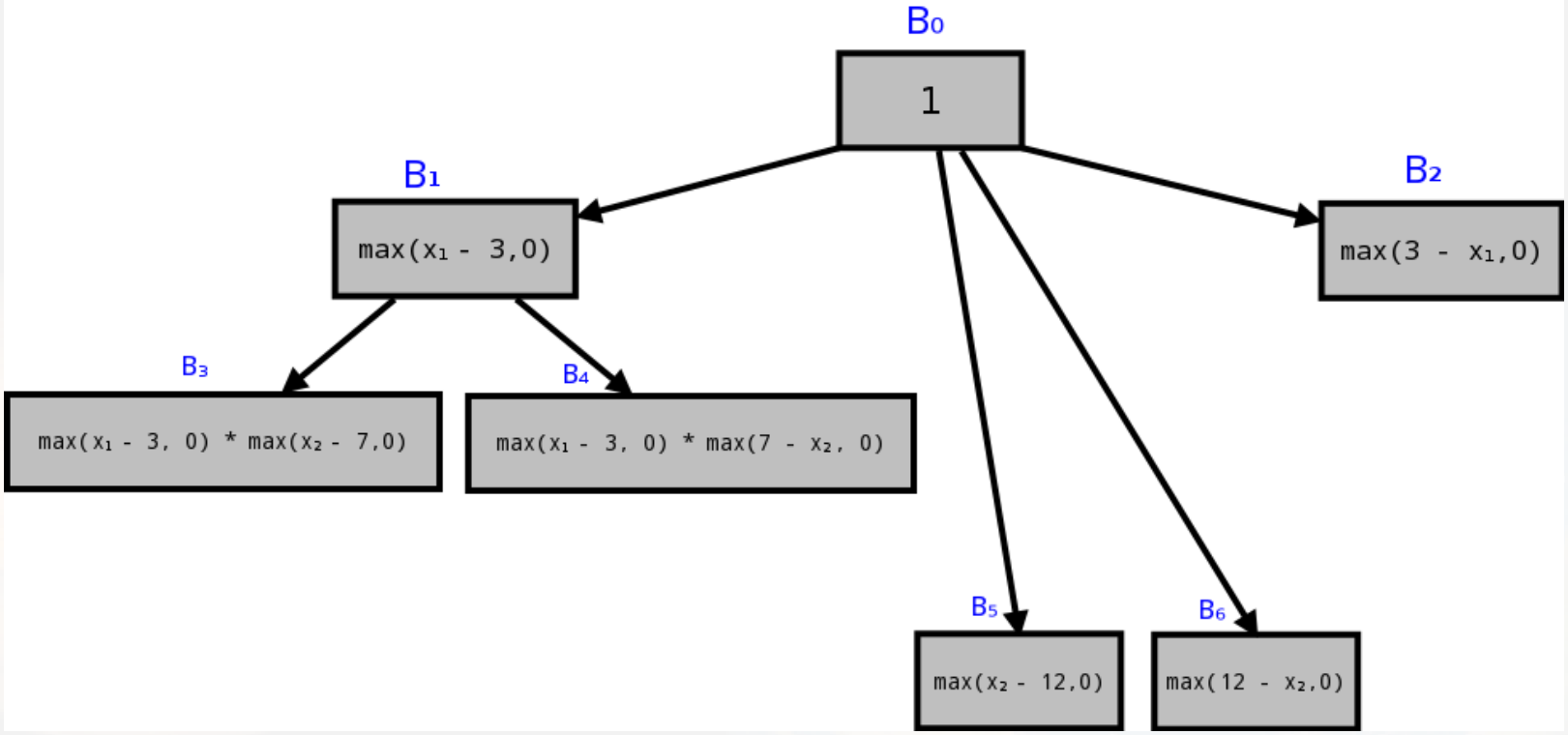
Copyright © Walter Myers

How does MARS work ? The forward pass



Copyright © Walter Myers

How does MARS work ? The forward pass



$$y = (\alpha_0 + \alpha_1 max(x_1 - 3, 0) + \alpha_2 max(3 - x_1, 0) + \alpha_3 max(x_1 - 3, 0) max(x_2 - 7, 0) + \alpha_4 max(x_1 - 3, 0) max(7 - x_2, 0) + \alpha_5 max(x_2 - 12, 0) + \alpha_6 max(12 - x_2, 0)) \quad (1)$$

Copyright © Walter Myers

What have been done ?

- **Github repo** : https://github.com/jcrudy/py-earth
  - `git clone https://github.com/jcrudy/py-earth`
  - `cd py-earth`
  - `python setup.py install`
- The state of the code:
  - **Py-earth** supported already a lot of features and the important parts were there
    - However, it was not ready to be deployable to `scikit-learn`
    - it was not supporting `multiple outputs`

Copyright © Walter Myers

What have been done ?

- **Github repo** : https://github.com/jcrudy/py-earth
  - `git clone https://github.com/jcrudy/py-earth`
  - `cd py-earth`
  - `python setup.py install`
- The state of the code:
  - **Py-earth** supported already a lot of features and the important parts were there
    - However, it was not ready to be deployable to `scikit-learn`
    - it was not supporting `multiple outputs`

Copyright © Walter Myers

What have been done ? improve code quality

- Clean the code (pep8) and adapt it to coding guidelines of `scikit-learn`
- Enchance documentation
- Add more unit tests

Copyright © Walter Myers

What have been done ? improve code quality

- Clean the code (pep8) and adapt it to coding guidelines of `scikit-learn`
- Enchance documentation
- Add more unit tests

Copyright © Walter Myers

What have been done ? improve code quality

- Clean the code (pep8) and adapt it to coding guidelines of `scikit-learn`
- Enchance documentation
- Add more unit tests

Copyright © Walter Myers

What have been done ? new features

- Support for `multiple outputs`
- Support for `output weights`
- Support of estimation of `variable importance`
- Implement `FastMARS` (Jerome H.Friedman, 1993)

Copyright © Walter Myers

What have been done ? new features

- Support for `multiple outputs`
- Support for output weights
- Support of estimation of `variable importance`
- Implement `FastMARS` (Jerome H.Friedman, 1993)

Copyright © Walter Myers

What have been done ? new features

- Support for `multiple outputs`
- Support for `output weights`
- Support of estimation of `variable importance`
- Implement `FastMARS` (Jerome H.Friedman, 1993)

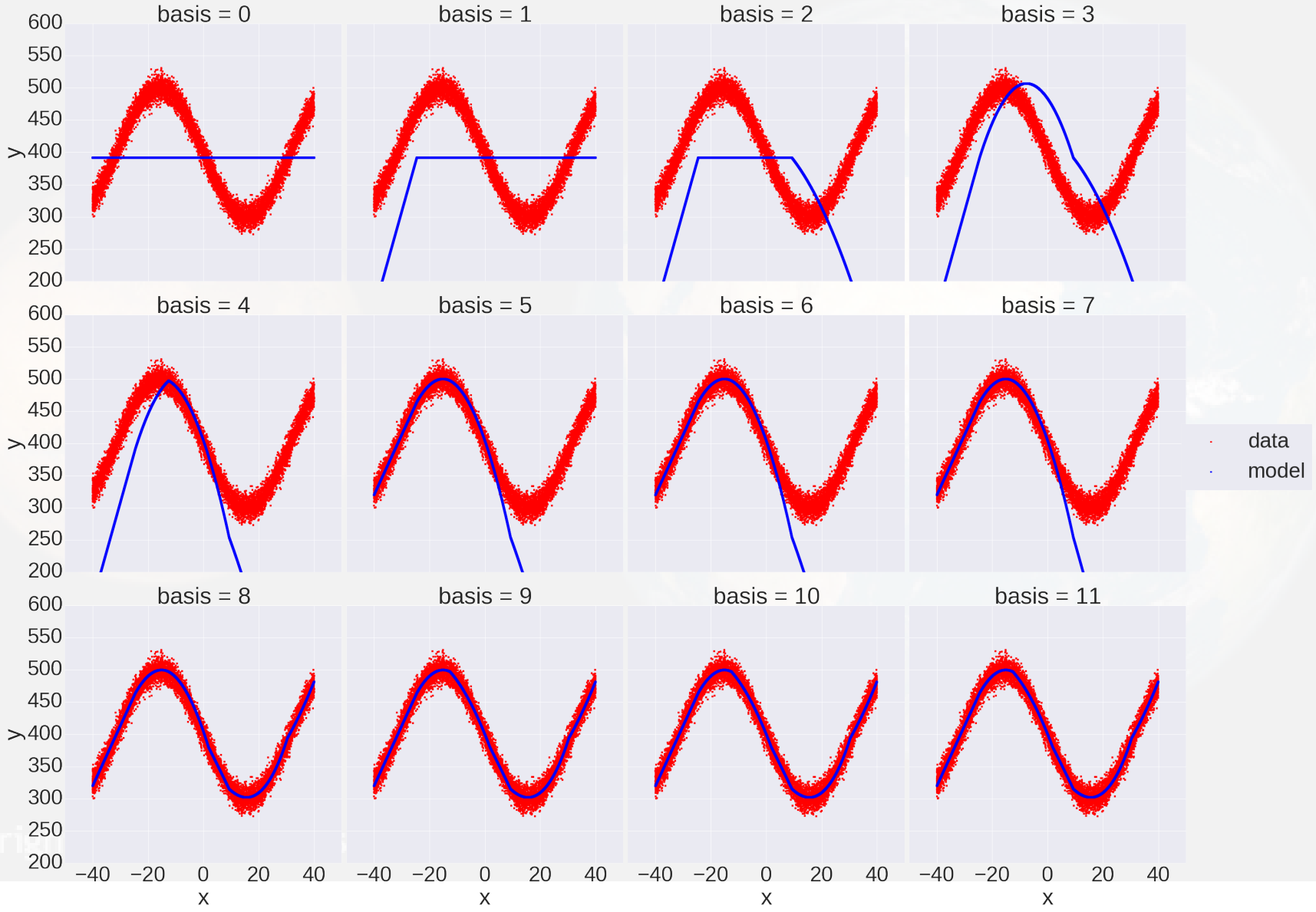
Copyright © Walter Myers

What have been done ? new features

- Support for `multiple outputs`
- Support for `output weights`
- Support of estimation of `variable importance`
- Implement `FastMARS` (Jerome H.Friedman, 1993)

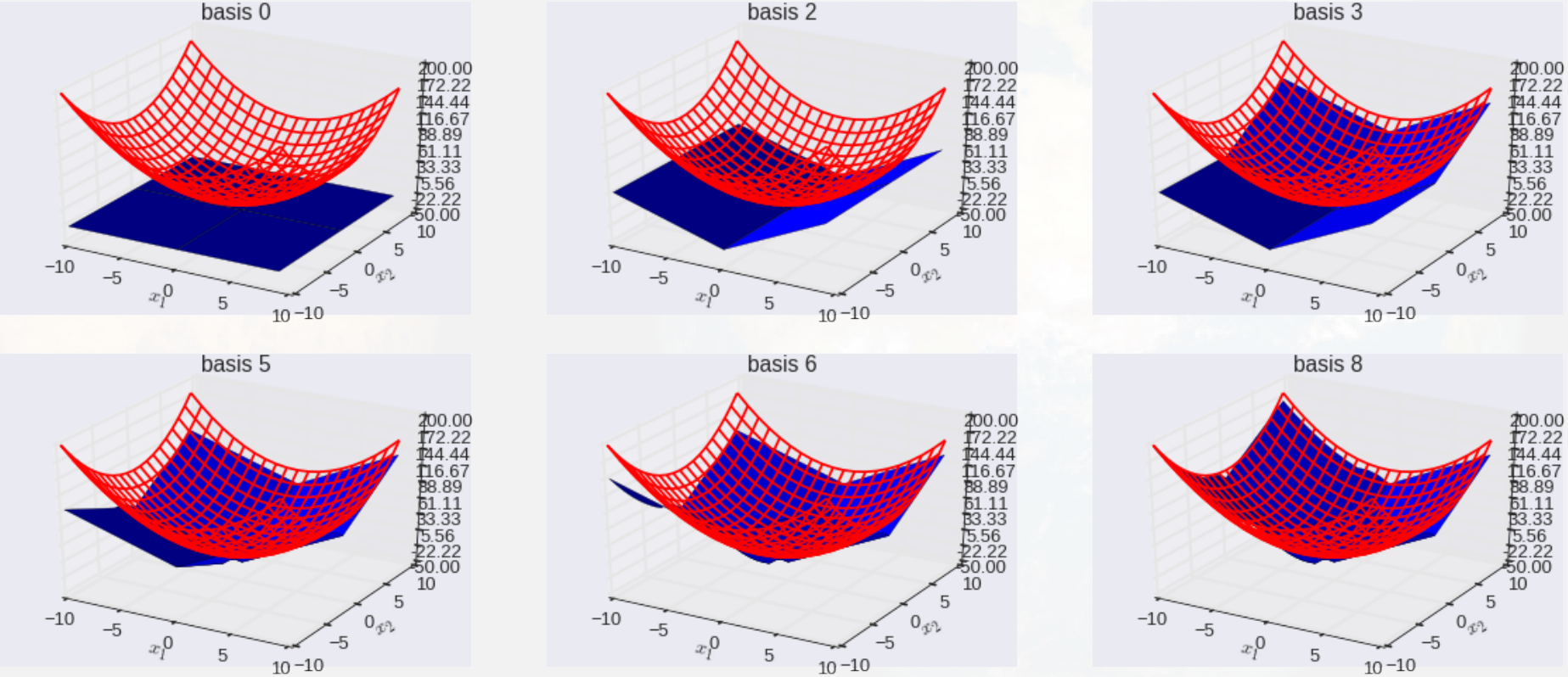
Copyright © Walter Myers

Example : 1D example



Copyri

Example : 2D example



Copyright © Walter Myers

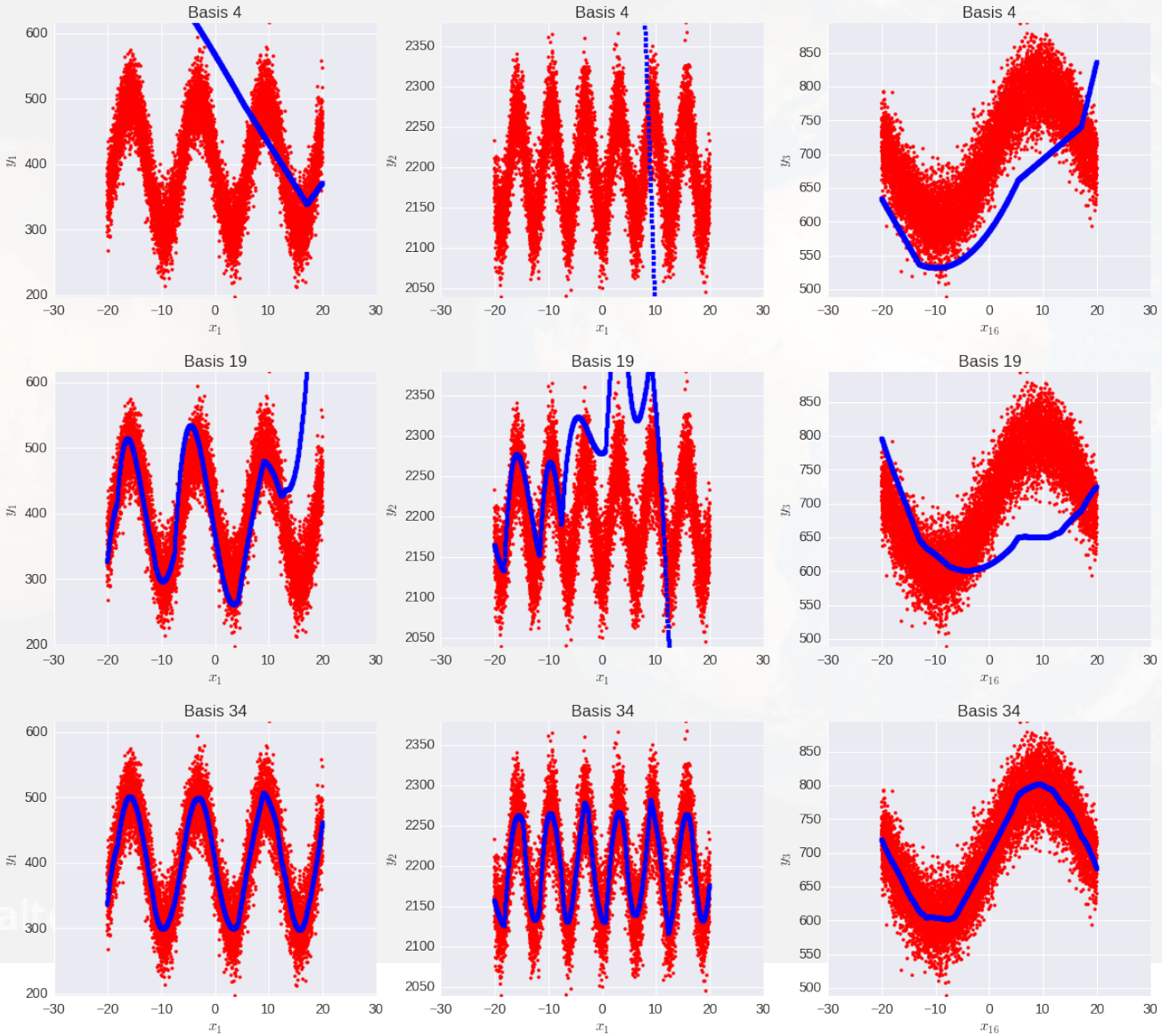
Example : multiple outputs

```
from pyearth import Earth
m = 10000
low, high = -20, 20
X = np.random.uniform(low, high, size=(m, 20))
y = np.zeros((m, 3))

y[:, 0] = (100 *np.abs(np.sin((X[:, 0]) * 0.5) - 4.0) +
           30 * np.random.normal(size=m))
y[:, 1] = (100 * np.abs(np.cos((X[:, 0]))*0.7 - 22.0) +
           30 * np.random.normal(size=m))
y[:, 2] = (100 * np.abs(np.sin((X[:, 15]) / 6) + 7.0) +
           30 * np.random.normal(size=m))
model = Earth(max_terms=40, max_degree=10)
model.fit(X, y)
print(model.summary())
```

Copyright © Walter Myers

Example : multiple outputs

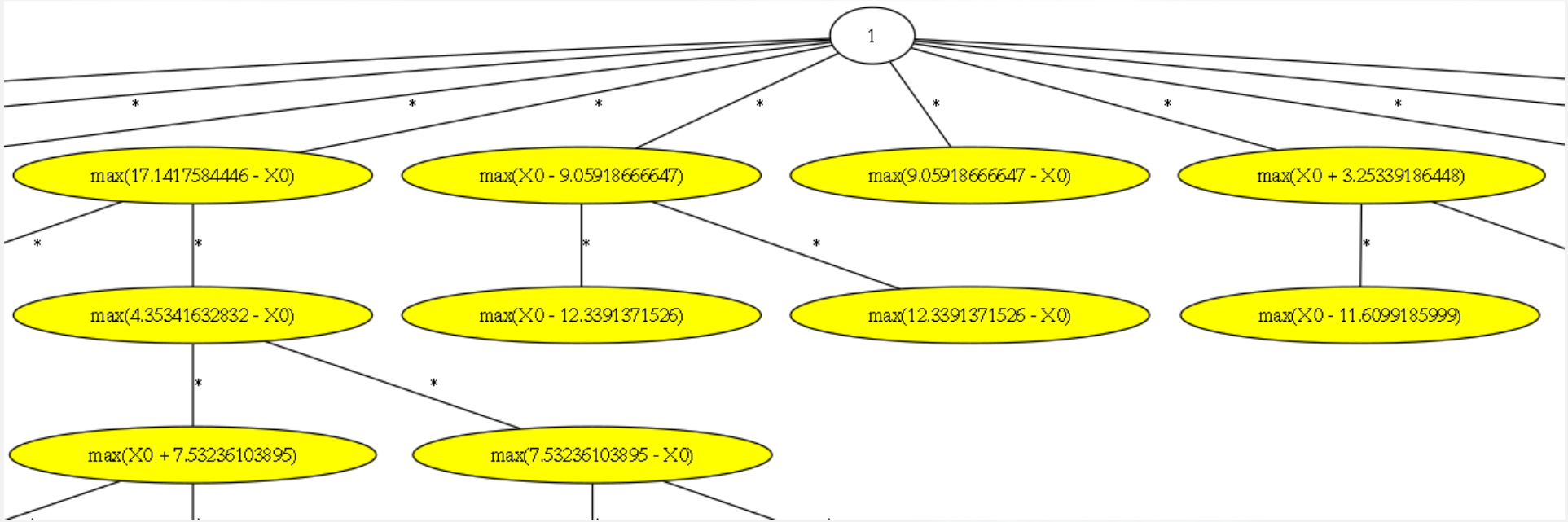
20 inputs, **3** outputs, only **2** informative inputs, the rest is noise



Copyright © Walt

Example : multiple outputs

The graph (a crop of it) of basis functions looks like this :



Copyright © Walter Myers

Example : variable importance

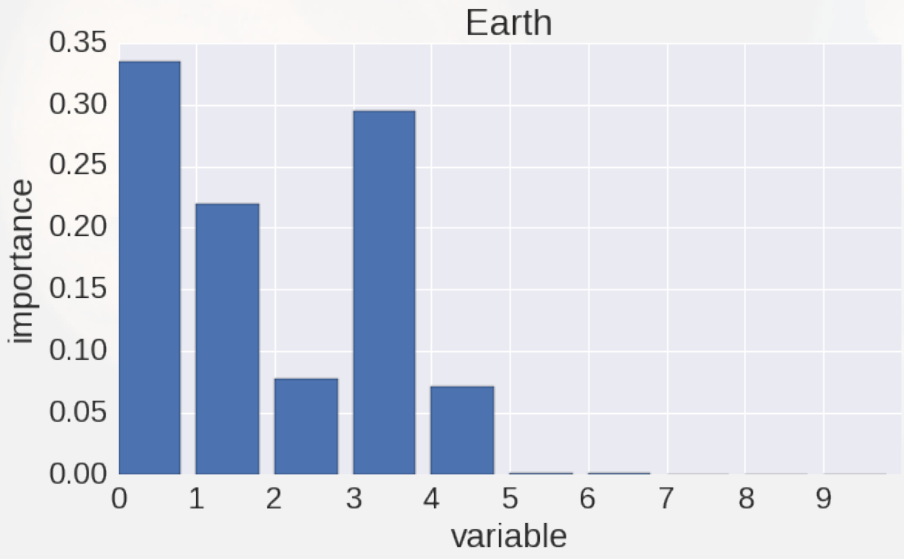
```
from sklearn.datasets import make_friedman1
from pyearth import Earth
from sklearn.ensemble import RandomForestRegressor

X, y = make_friedman1(n_samples=1000, noise=5, random_state=123456)

earth = Earth(max_degree=10, feature_importance="gcv")
earth.fit(X, y)

rf = RandomForestRegressor(n_estimators=300)
rf.fit(X, y)

inputs = np.arange(X.shape[1])
fig = plt.figure(figsize=(20, 5))

plt.subplot(1, 2, 1)
plt.bar(inputs, earth.feature_importances_)
plt.title("Earth")
plt.xticks(inputs)
plt.xlabel("variable")
plt.ylabel("importance")

plt.subplot(1, 2, 2)
plt.bar(inputs, rf.feature_importances_)
plt.title("RandomForestRegressor")
plt.xticks(inputs)
plt.xlabel("variable")
plt.ylabel("importance")
```

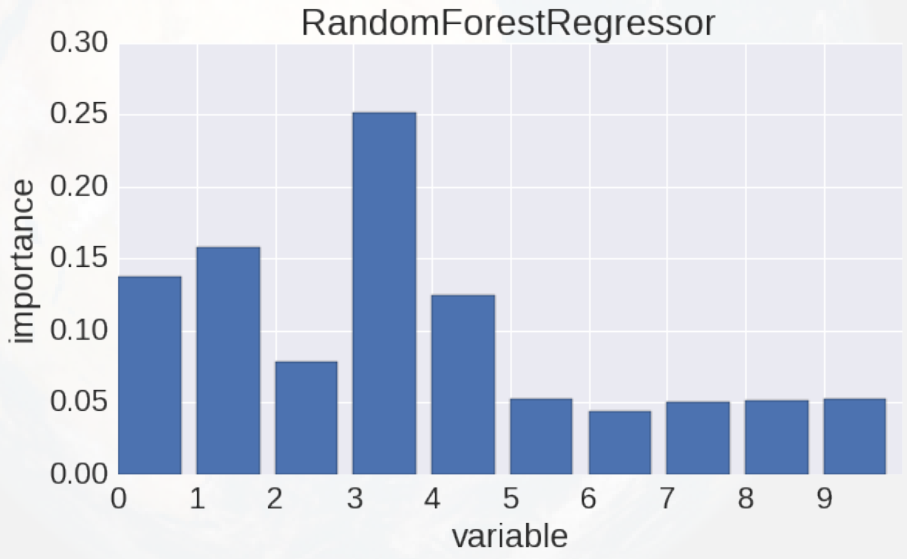
Copyrigh

Example : variable importance

$y = sin(\pi x_0 x_1) + 20(x_2 - 0.5)^2 + 10x_3 + 5x_4 + 5 * N(0,1)$

The code in the previous slide gives:





Copyright © Walter Myers

Example : FastMARS

```
from pyearth import Earth
m = 10000
low, high = -20, 20
X = np.random.uniform(low, high, size=(m, 20))
y = np.zeros((m, 3))

y[:, 0] = (100 *np.abs(np.sin((X[:, 0]) * 0.5) - 4.0) +
           30 * np.random.normal(size=m))
y[:, 1] = (100 * np.abs(np.cos((X[:, 0]))*0.7 - 22.0) +
           30 * np.random.normal(size=m))
y[:, 2] = (100 * np.abs(np.sin((X[:, 15]) / 6) + 7.0) +
           30 * np.random.normal(size=m))
```

```
%%time
model_slow = Earth(max_terms=40, max_degree=12)
model_slow.fit(X, y)
print("MSE : {0}".format(model_slow.mse_))
```

```
MSE : 8101.20518621
CPU times: user 4min 17s, sys: 3min 13s, total: 7min 31s
Wall time: 6min 50s
```

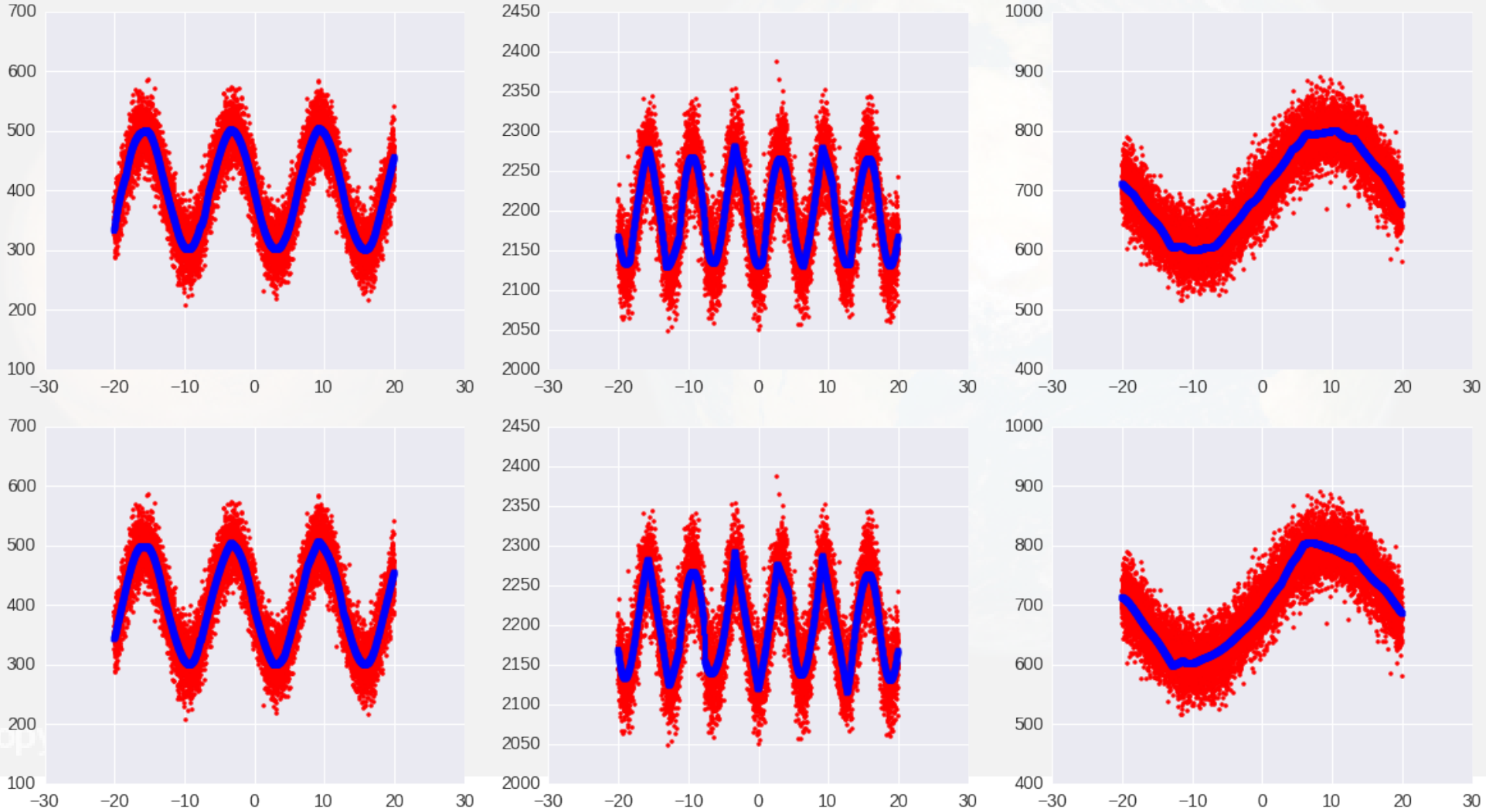
```
%%time
model_fast = Earth(max_terms=40, max_degree=12, use_fast=True, fast_K=8, fast_h=1)
model_fast.fit(X, y)
print("MSE fast : {0}".format(model_fast.mse_))
```

```
MSE fast : 8301.02747927
CPU times: user 59.2 s, sys: 50.5 s, total: 1min 49s
Wall time: 1min 10s
```

Copyr

Example : FastMARS

Top : Normal, Bottom : FastMARS



Future

- Close current issues, keep working on code quality to merge it into scikit-learn
- Still, some features are missing, new features:
  - Deal with missing values
  - Support categorical variables

Copyright © Walter Myers

Future

- Close current issues, keep working on code quality to merge it into scikit-learn
- Still, some features are missing, new features:
  - Deal with missing values
  - Support categorical variables

Copyright © Walter Myers

Future

- Close current issues, keep working on code quality to merge it into scikit-learn
- Still, some features are missing, new features:
  - Deal with missing values
  - Support categorical variables

Copyright © Walter Myers

Thank you

Thank you for listening

Copyright © Walter Myers