Zhaoyang Wan

Using Python to Develop Analytics, Control and Machine Learning Products

Using Python to Develop Analytics, Control and Machine Learning Products

Zhaoyang Wan

Dedicated to my wife, who devotes herself to our family; to my kids Charley and Richard, who make me a kid at heart; to my parents, who constantly use their time-tested wisdom to lighten my path ahead.

Contents

- Why to write this book
- Basics
  - The Law of Effect
  - The Principle of Least Action
  - Python - The Category Theory
  - Additional Lessons learnt
  - An Example of Simple Analytics Product
  - Installation of Python
  - Core Python: Set Operations
  - Core Python: Iteration Functions
  - Core Python: Lambda function
  - Core Python: Dictionary Data Structure and Operations
- A Simple Analytics Product
  - The Architecture
  - Web/File Exchange/Ipython Notebook Servers
  - Web User Interface
  - Data Collection
  - Database and Asset Model
  - Turning Data into Information
  - Analytics
    - Single Variable
    - Multi-Variables
  - Report /Email/Survey
  - Simulation and Python Class
  - Deployment to Cloud
  - Collaboration on Github or Ipython Notebook and FTP Server
- An Observe-Identify-Diagnose-Improve Analytics Product
  - The architecture
  - Web User Interactive Interface

- Report with 3D and Animation
- Analyze, Quantify Risk and Tradeoff Return and Risk
- Modular Approach

A Supervisory Modeling, Optimization and Learning Product

- The Architecture
- User Interface
- Communications
  - Serial and Parallel interfaces
  - OPC Server and Client
- Modeling
  - Linear Regression
  - Learning Nonlinearity: Spline Interpolation
  - Learning Nonlinearity: Classification
  - Learning Nonlinearity: Multivariate Adaptive Regression Splines (MARS)
  - Learning Nonlinearity: Neural Network
  - Keep Big Picture: Nonlinear Transformation
  - Keep Big Picture: Surrogate Model
  - Keep Big Picture: Model Reduction
- Balance between Forward Induction and Backward Induction
  - Application 1: Recommendation Engine
  - Application 2: Reinforcement Learning
  - Application 3: Convex Optimization and Global Optimization
  - Application 4: Model Identification, Optimal Control and State Estimation

Next Generation Analytics: Analytics with “Free Will”

Summary

- Product or Service
- Open or Close
- Technology and Humanity
- Impact Analytics Product Development for the Better
- Are We Better off with Internet of Things?
- Instead, Call It Analytic Field
- Bring in Stochastics and Uncertainty, Exploration and Learning

This is the Way to Ride Next Internet Revolution…

Why to write this book

Firstly, I write this book for my kids. My kids are in 6th grade and 2nd grade, they have a habit of writing. I want to be a role model for them to show them that writing should be a lifelong habit.

Secondly, I write this book for my parents. My parents are in their 80's. Both were university professors before they retired. From their careers, they took great pride and satisfaction in transforming students into knowledgeable professionals. Each year at New Year's Day, their students would come to or call my parents and express their appreciations. My dad wrote many textbooks and got state honored awards for teaching excellence. He frequently advises me to summarize my learning daily and consolidate the learnings into books, which may have impacts for generations to come.

Thirdly, to continue my second point, I write this book for my fellow colleagues in system engineering. Living in US, we are facing global competitions, I found out one way of gaining competitiveness is to develop the capability of building an integrated product, which integrates software, hardware and engineering together. Moreover, as we grow older and build wealth, it is imperative for us to maintain the lead by developing deeper understanding of the world and building more powerful technology.

Fourthly, as I am writing various knowledge down and explaining them to myself, I realize various knowledge in fact have a lot in common, and they are just approaching to the same problem from different angles (e.g. optimal control, reinforcement learning and dynamic game theory). I feel like that there is "an invisible hand" (much like the "invisible hand" Adam Smith used in his Wealth of Nation to describe market economy in balancing market supply and demand) that balances various approaches in the problem-solving process.

Indeed, solving a problem is like playing a game with mother nature, a solution can be derived by either forward induction or backward induction or balance of the two (e.g. forward induction to identify model and estimate unmeasured variables, and backward induction to decide optimal action based on identified model and measured/estimated variables)

"Backward induction *represents a pattern of reasoning in which a player, at*

every stage of the game, only reasons about the opponents' future behavior and beliefs, and not about choices that have been made in the past. So, he takes the opponents' past choices for granted, but does not draw any new conclusions from these.

In contrast, ***forward induction*** *requires a player, at every stage, to think critically about the observed past choices by his opponents. He should always try to find a plausible reason for why his opponents have made precisely these choices in the past, and he should use this to possibly reconsider his belief about the opponents' future, present, and unobserved past choices."* (http://www.mdpi.com/2073-4336/1/3/168/pdf)

One example is that my son usually prefers "trial and error" to solve word problems (a forward induction approach), while I prefer setting up unknowns and solving the equations representing the word problems (a backward induction approach). For another example, an optimum of a function can be solved analytically by solving its first order derivative equals to zero (a backward induction approach), or solved numerically by a natural selection process that mimics biological evolution (a forward induction approach).

Lastly, I write this book for myself. It is "amazing" how fast we forget what we learn. Our learning is like drawing in the sand on the beach, which will be quickly washed away by the waves. Writing it down and explaining it to myself is one way to ensure a lasting memory. In fact, the more I write, the more I crave to collect those scattered intellectual "treasures" into my treasure chest. To bring more contents into this book, I provide many web sites in the text so that this summary book can be linked to more detailed knowledge.

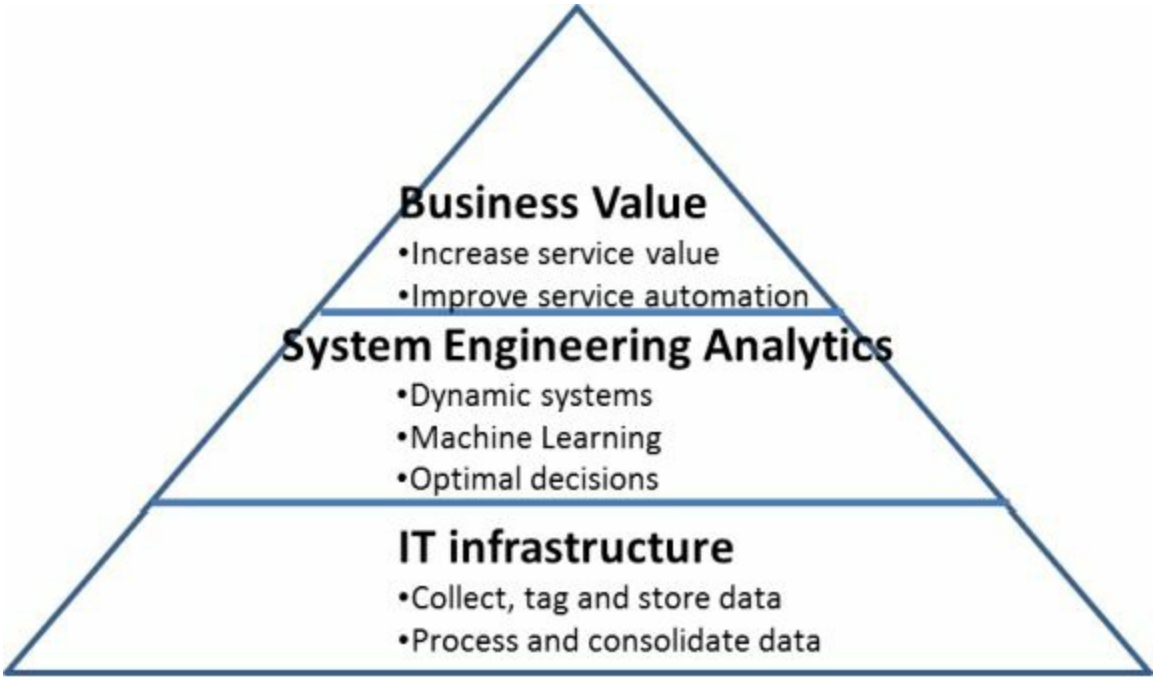
Basics

The Law of Effect

In a business, after data, related to the products the business is selling, is collected, product support looks into the data to resolve product issues and develop better products.

The Business Analytics changes the game by pushing product data to the enterprise level to allow businesses to leverage big data to optimize processes and asset performance, apply predictive analytics to minimize unplanned downtime, increase throughput, improve product quality, and drive resource efficiency.

Business analytics is an integrated system, which consists of IT infrastructure and System Engineering Analytics to provide service value and service automation. It is critical for customer retention, and more importantly, it is service differentiator to gain more customers.



At a technical level, any business analytics is a search algorithm over the state space of different strategies in order to maximize a business value function. The search is done based on some combination of exploration and exploitation, which means whether to take the strategy that gave the highest value last time we were in this state, or trying out a different strategy in the hope of finding

something even better. The underlying principle is the Law of Effect. The essential idea is that behavior can be modified by its consequences, i.e.

*"responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation." (*https://en.wikipedia.org/wiki/Law_of_effect *)*

The very nature of data driven implies any business analytics is well grounded in the Law of Effect.

The Principle of Least Action

"Nature is thrifty in all its actions". The principle that describes this is called the principle of least action (https://en.wikipedia.org/wiki/Principle_of_least_action). Basically, it states that Nature always finds the most efficient course from one point to another. This principle remains central in modern physics and mathematics, being applied in the theory of relativity and quantum mechanics (https://www.youtube.com/watch?v=IhlSqwZBW1M). A simple example of such principle is to find the curve of shortest length connecting two points. If there are no constraints, the solution is obviously a straight line between the points.

The idea of "One-line" business analytics, meaning connecting data to business value in "a straight line", is inspired by this principle, with the goal to make analytics development as efficient as possible and make analytics applications as easy as possible.

Indeed, human world shares the same thriftiness character with Nature. The principle of least effort postulates that animals, people, even well designed machines will naturally choose the path of least resistance or "effort".

Steve Jobs once said, "*We human (the crown of creation) are tool builders, and we can fashion tools that amplify the inherit ability we have to spectacular magnitude, and take us far beyond our inherit ability*." This is because we human constantly apply Nature's principle to achieve maximum business value with minimum efforts in learning, analyzing and decision making. For example, powerful computer languages are invented to achieve minimalistic programming, which in turn dramatically simplify the task of developing an end-to-end business analytics product. Once you discover that there is a simple straight line towards building a powerful business analytics product, your world will never be the same again.

Python - The Category Theory

1. *Everything in Python is an object.*
2. *Any object in Python can be grouped into a set.*
3. *Any set in Python can be transformed to any other set via some functions.*

The logical deduction of the above statements is

"Anything in Python can be transformed into anything else by simply applying functors to sets of objects". (https://www.youtube.com/watch?v=u2KZJzoz-qI)

This conclusion simply means you can pack the start set, the transformation functions and the target set into One Line. In fact, conceptually, the entire business analytics product can be written in one line:

```
>>> WebService().Data().Preprocess().Analyze().Visualize().Store().Automate()
```

Moreover, entire Python ecosystem carries out this same basic principle and makes many seemingly complex transformations into simple, smart and intuitive One-Liners.

This high level of abstraction of Python is inspired by category theory (https://en.wikipedia.org/wiki/Category_theory). Category theory is an abstraction of mathematics itself that allows many significant areas of mathematics can be formalized by category theory as categories, i.e. collections of objects and arrows (also called morphisms). Functors are structure-preserving maps between categories. In our case, we can think each step of the entire business analytics system as a functor, which transforms one category to the other category.

Additional Lessons learnt

- **Be an integrator**. As many "physical" products, once proprietary, become commoditized, services, integrating domain expertise around products, become the next generation differentiator. To make this happen, a system engineer needs to integrate many critical interfaces to channel data and expertise into services and deliver them to customers. Only after deploying the entire integration, can service value be realized.

  A business analytics system itself is a "digital" product that we are putting into the outside world, seeking to integrate yet differentiate from other business service work flows. However, in this highly specialized technical world, we tend to think of how to reduce the problem into the field we are specialized in, not thinking of expanding it to a fully functioning business solution, including both integration of system engineering expertise (e.g. optimization, machine learning, dynamics system & control and visualization) and integration of data collection, pre-processing and service automation.

- **Be abstract**. The ability to go broader in integration is based on the ability to go deeper in abstraction. Otherwise, you will be buried by the seemingly complex system. Fortunately, we have math and more importantly Nature to provide a deeper level of abstraction to capture real world complexity. Don't be afraid, the world at its deepest abstraction makes perfect common sense. Aren't we all inspired by the way Nature governs our universe?

  The laws of movement and of rest deduced from the principle (of least action) being precisely the same as those observed in nature, we can admire the application of it to all phenomena. The movement of animals, the vegetative growth of plants ... are only its consequences; and the spectacle of the universe becomes so much the grander, so much more beautiful, the worthier of its Author, when one knows that a small number of laws, most wisely established, suffice for all movements. -Pierre Louis Maupertuis (https://en.wikipedia.org/wiki/Principle_of_least_action)

- **Be simple and intuitive**. Simplicity drives service adoption. In fact, "service" by its very definition ("the action of helping or doing work for someone") implies giving someone peace of mind. The ability of being simple and intuitive is based on seamless integration and high level of abstraction conveyed in forms of common senses, e.g. an intuitive visualization can convey information at one glance, better than speaking a

thousand words.

- **Be automated**. Analytics service should not be "pulled" by customers. It should be "pushed" to customers automatically (sometimes even without customers noticing them). In order to push analytics to all customers, common structure among them needs to be extracted by dissecting the world into the same "bag of words" and "bag of relationships"
- **Be real time and predictive**. Probably there is not much value left if knowledge gained in the service is for something that happened in the past, but it will offer tremendous value if the knowledge gained is channeled into real time decision-making, and even more valuable if it can help optimize the future.
- **Be open**. Seeing how vibrant the open source IT world is, you would sure want to open up your world to your colleagues with the shared vision and mission, and contribute and collaborate to make it a better shared community.

An Example of Simple Analytics Product

1. **Set up IPython Notebook Server (**https://ipython.org/ipython-doc/1/interactive/public_server.html **), run Server at** http://yourIP:9999/tree

C:>ipython profile create nbserver
C:>ipython notebook --profile=nbserver

2. **Design web user interface (https://www.youtube.com/watch?v=VaV10VNZCLA), WebService()**

from IPython.html.widgets import *
interact((lambda a,t,analyze: analytics(a,t,analyze)), a=['asset1','asset2','asset3'],t=(0,10),analyze=False)

```
from IPython.html.widgets import *
interact((lambda a,t,analyze: analytics(a,t,analyze)), a=['asset1','asset2','asset3'],t=(0,10),analyze=False)
```

3. **One line Python, Data().Preprocess().Analyze().Visualize().Store().Automate()**

The analytics automatically collects time series data from a data file, removes time series with all nan and constant values, resamples data at hourly interval, removes data records with nan values, performs correlation analysis and stores correlation coefficients in database.

 # Data()

```
from tables import *
f=open_file('DB.h5','w')
f.create_table('/', 'Analytics', dtype([('var1', 's50'), ('var1', 's50'), ('corrcoef', 'float32')]));
f.close()
```

 # Preprocess().Analyze().Visualize().Store()

```
from pylab import *
from pandas import *
def Analytics():
    df=csv_read('data.csv',index_col=0,parse_dates=True);
```

```
    df=df.loc[:,df.apply(Series.nunique) != 0]\
       .loc[:,df.apply(Series.nunique) != 1]
       .resample('H',fill_method='ffill').dropna();
   R = corrcoef(df.T); pcolor(R); colorbar(); clim(-1,1);

    DB=open_file('Analytics.h5','a');
    Tab=DB.root.analytics; tab=Tab.row;
    for r in [(df.columns[i],df.columns[j],R[ii][jj]) for i in \
            range(len(R)) for j in range(i)]:
      tab['var1']=r[0];
      tab['var2']=r[1];
      tab['corrcoef']=r[3];
      analytics.append()
   Tab.flush(); DB.close()
```

 # Automate()

```
import datetime, time
while True:
    Analytics() if datetime.datetime.now().minute==0 else time.sleep(60)
```

Installation of Python

There are 3 major scientific Python distributions:

- **Anaconda** (http://continuum.io/downloads)
- **Canopy** (https://store.enthought.com/)
- **Pythonxy** (https://code.google.com/p/pythonxy/)

Although all three distributions have most of the popular site packages, they each have their uniqueness, e.g.

- Pythonxy has the most packages in its distribution, specifically the optimization package "cvxopt" is distributed only with pythonxy
- Canopy has the most integrated programming, package manager, document and training environment.
- Anaconda is the most compact distribution; the web plotting package "bokeh" is developed by Continuum and distributed with Continuum Anaconda.

Core Python: Set Operations

Core Python feature is best captured by the statement "*Anything in Python can be transformed into anything else by simply applying functions to sets of objects*". This is the deepest abstraction of our interactions with the real world.

It is worthwhile to take a moment to understand the underlying structure, supporting this all-encompassing abstraction.

- Python set operations that describe relationship between groups of objects
- Python function and iteration operations that apply function iteratively through each element of the set

In fact, the underlying structure supports the basic operation of our mind, i.e. discovering patterns and cause-and-effect relationship through repetition.

Union

```
>>> a=set([1,2,3,4,5])
>>> b=set([3,4,5,6,7])
>>> a|b #show the union of set a and b
set([1, 2, 3, 4, 5, 6, 7])
```

Intersect

```
>>> a&b #show the intersect of set a and b
set([3, 4, 5])
```

Difference

```
>>> a-b #show the elements in set a but not in b
set([1, 2])
>>> b-a #show the elements in set b but not in a
set([6, 7])
>>> a^b #show the elements in set a but not in b or in set b but not in a
set([1, 2, 6, 7])
```

Core Python: Iteration Functions

```
>>> from itertools import *
>>> c=count(1,2)
>>> print c.next(),c.next(),c.next()
1 3 5
>>> c=cycle('ABC')
>>> print c.next(),c.next(),c.next(),c.next()
A B C A
>>> list(repeat(10, 3))
[10, 10, 10]
>>> from itertools import *
>>> x=['a','b','c']
>>> y=['d','e','f']
>>> x[0]
'a'
>>> zip(x,y)
[('a', 'd'), ('b', 'e'), ('c', 'f')]
>>> list(chain(x,y))
['a', 'b', 'c', 'd', 'e', 'f']
>>> list(product(x,y))
[('a', 'd'), ('a', 'e'), ('a', 'f'), ('b', 'd'), ('b', 'e'), ('b', 'f'), ('c', 'd'), ('c', 'e'), ('c', 'f')]
>>> list(combinations(x,2))
[('a', 'b'), ('a', 'c'), ('b', 'c')]
>>> list(permutations(x,2))
[('a', 'b'), ('a', 'c'), ('b', 'a'), ('b', 'c'), ('c', 'a'), ('c', 'b')]
>>> list(compress('ABCDEF', [1,0,1,0,1,1]))
['A', 'C', 'E', 'F']
```

Core Python: Lambda function

```
>>> f=lambda x:x*2
>>> f(4)
8
>>> i=[1,2,3,4,5,6]
>>> filter((lambda x: x%2==0),i)
[2, 4, 6]
>>> map((lambda x:x*2),i)
[2, 4, 6, 8, 10, 12]
>>> reduce((lambda x,u: x+u),i)
21
```

In these three functions, the first parameter is a function, the second parameter is a list input, and the function is used to filter, map or reduce the list.

These are perfect examples for the statement "*Anything in Python can be transformed into anything else by simply applying functions to sets of objects*". "filter" and "map" are static transformation (which, when applied to each item of the list, only depends on that item), while "reduce" is dynamic transformation (which, when applied to each item of the list, depends on that item and all the items prior to it). In general, any transformation can be decomposed into a combination of a set of static and dynamic transformations.

Core Python: Dictionary Data Structure and Operations

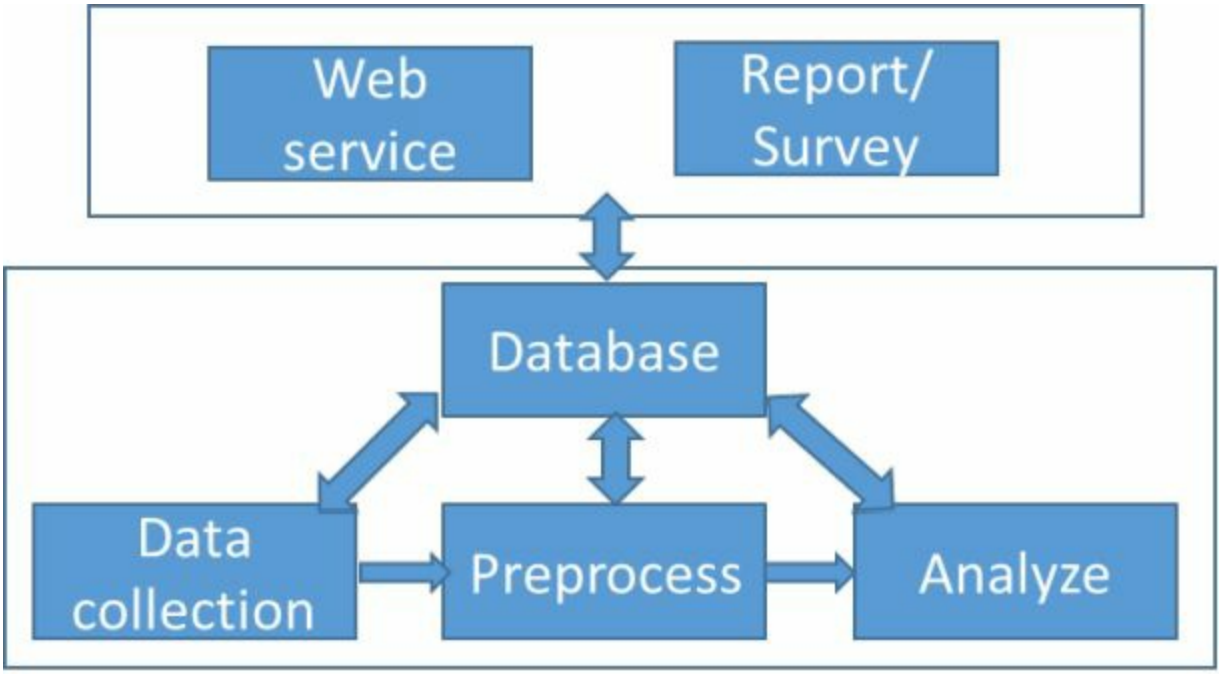
```
>>> y={'a':1,'b':2,'c':3,'d':4,'e':5}
>>> y['a'] #dictionary element is referenced by its label
1
```

Python Pandas package (http://pandas.pydata.org/) further extends Dictionary operations to Series and DataFrame operations.

```
>>> from pandas import *
>>> s=Series([1,2,3,4,5],index=['a','b','c','d','e'])
>>> df=DataFrame({'one':Series([1,2,3,4,5],index=['a','b','c','d','e']),...
'two':Series([6,7,8,9,10],index=['a','b','c','d','e'])})
>>> df
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
```

A Simple Analytics Product

The Architecture



Web/File Exchange/Ipython Notebook Servers

Providing web service is an obvious choice, because users do not need to install any client side software, which means easy access for users, plus it provides easy update and protection of software at the server side.

A web service includes two parts:

- A web interface, which provides users with intuitive forms to fill out their requests
- A server, which responses to users' requests

This involves two transformations

- A transformation from visual selection in web interface to web service requests
- A transformation from web service requests to execution of services corresponding to the requests

What Python does is making the above two transformations extremely simple, which mostly can be carried out in one line.

Simple HTTP Server

In Command Window, run

```
python -m SimpleHTTPServer
```

CGI HTTP Server

In Command Window, run

```
python -m CGIHTTPServer
```

FTP Server

Download pyftpdlib and in Command Window, run

```
python -m pyftpdlib
```

IPython Notebook Web Server

In Command Window, run

```
ipython profile create nbserver
ipython notebook --profile=nbserver
```

Web User Interface

There are many python web development templates/frameworks. There are 2 minimalist ones, i.e. Bottle and Flask.

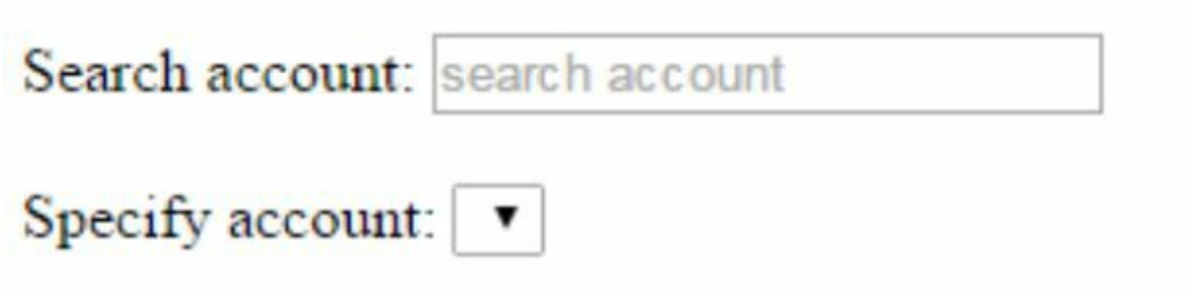
Bottle (http://bottlepy.org/docs/dev/ **)**

```
from bottle import route, run, template
@route('/hello/<name>')
def index(name):
   return template('<b>Hello {{name}}</b>!', name=name)
run(host='localhost', port=8080)
```

Flask (http://flask.pocoo.org/ **)**

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
   return "Hello World!"
if __name__ == "__main__":
   app.run()
```

Both share the common syntax. Both use "@" to define routing of web addresses, which may take web client side requests and then use python server side to respond to those requests, as shown in the following example in Bottle, which uses text search to identify account options in the selection.



Client side

```
<!DOCTYPE html>
<html lang="en">
<head>
<script type="text/javascript">
$(document).ready(function() {
  $('#search').keypress(
```

```
   function(e){
    if (e.keyCode==13) {
      $('#acctList').empty();
      $.ajax({
        url: '/acctList/'+$("#search").val(),
        cache:false,
        type: 'GET',
        success: function(data) {
        $('#acctList').append($('<option>').text("select").val("select"));
       $.each(data.result, function(index, value) {
        $('#acctList').append($('<option>').text(value).val(value));
        });
  }
});
     }
   });
  });
</script>
</head>
<body>
   <label for="search">Search account: </label>
   <input id="search" type="text" placeholder = "search account"/><br><br>

   <label for="acctList">Specify account: </label>
   <select name="acct" id="acctList"></select><br><br>
</body>
</html>
```

Server side

```
@route('/acctList/<inputval>', method='GET')
def acct(inputval):
  #define accList
  return {'result':acctList}
```

Data Collection

The world is connected by IP addresses. We can get data from anywhere, as long as data has entered into the IT world in any way via an IP address.

HTTP server

From HTTP server in a data collection device

```
import urllib
dataFile = urllib.urlopen('http://1.1.1.1:2121/dataFile.csv').read()
f = open('dataFile.csv', 'wb')
f.write(dataFile)
f.close()
```

FTP server

From FTP server in a data collection device

```
import urllib
dataFile = urllib.urlopen('ftp://1.1.1.1:8000/dataFile.csv').read()
f = open('dataFile.csv', 'wb')
f.write(dataFile)
```

Email server

From Email server in a data collection device

```
from win32com.client import Dispatch
outlook = Dispatch("Outlook.Application").GetNamespace("MAPI")
inbox = outlook.GetDefaultFolder("6")
all_inbox = inbox.Items
sub = 'data'
att = 'data.csv'
for msg in all_inbox:
    for att in msg.Attachments:
        if sub in msg.Subject and att in att.FileName:
            att.SaveAsFile('data_received.csv')
            att.ExtractFile('data_received.csv')
            open(att)
            att.WriteToFile(msg.Subject+att.FileName)
```

SQL server

From SQL server in a data collection device

```
import cx_Oracle
con = cx_Oracle.connect("<IP address>")
cur = con.cursor()

cur.execute('''<SQL query>  ''', v1=c,v2=ind)
st=cur.fetchall()
```

Web server

From web server in a data collection device

```
import urllib
import json

serviceurl = 'http://maps.googleapis.com/maps/api/geocode/json?'

address = raw_input('Enter location: ')
url = serviceurl + urllib.urlencode({'sensor':'false', 'address': address})
print 'Retrieving', url
uh = urllib.urlopen(url)
data = uh.read()
```

Database and Asset Model

PyTables (http://www.pytables.org/) is a package for managing hierarchical datasets and designed to efficiently and easily cope with extremely large amounts of data. PyTables is built on top of the HDF5 library, using the Python language and the NumPy package. It features an object-oriented interface that, combined with C extensions for the performance-critical parts of the code (generated using Cython), makes it a fast, yet extremely easy to use tool for interactively browse, process and search very large amounts of data. One important feature of PyTables is that it optimizes memory and disk resources so that data takes much less space (specially if on-flight compression is used) than other solutions such as relational or object oriented databases.

Setup Database and Asset Model

```
from tables import *
f=open_file('Database.h5','w')
class Table(IsDescription):
   par1 = StringCol(200);
   par2 = Float32Col()
f.create_table('/','Table',Table)
f.close()
```

Open database and get and store data

```
DB=open_file('Database.h5','a')
Tab=DB.root.Table;
dat=Tab.row;
dat['par1']= 'par1'
dat['par2']= 80
dat.append()
Tab.flush();
DB.close()
```

You can use **pandas hdf read/write API** to access database, for example

```
df_tl = pd.DataFrame(dict(A=list(range(5)), B=list(range(5))))
df_tl.to_hdf('store_tl.h5','table',append=True)
pd.read_hdf('store_tl.h5', 'table', where = ['index>2'])
```

The table can also be view in ViTable (http://vitables.org/)

The above is the conventional way of using database from IT perspective. Engineers normally look at database more like a storage, so it is also very convenient and easy to use **pandas HDFstore (**http://pandas.pydata.org/pandas-docs/stable/cookbook.html#hdfstore **)**, for example,

```
store = pd.HDFStore('store.h5')
store['s'] = s #s is Series data
store['df'] = df #df is DataFrame data
store['wp'] = wp #wp is Panel data
del store['wp']
store.close()
```

This helps free up engineers' mind from setting rigid data structure beforehand, and allows engineers to gradually evolve and form the data structure through iterations. Changing database structure is a headache for IT, yet it is necessary for engineers to explore the solution space.

Indeed, analytics is all above data structure, we should let it evolve into a highly abstract and powerful data structure, which can save a thousand lines of code.

Turning Data into Information

In this digital world, data is gold mine, and the power of the mind can turn it into gold. With Pandas package (http://pandas.pydata.org/), one would be able to

- **Refer to any element of the table.**

```
>>> df=DataFrame({'c1':Series([1,2,3],index=
['i1','i2','i3']),'c2':Series([4,5,6],index=['i1','i2','i3'])})
>>> df
   c1  c2
i1   1   4
i2   2   5
i3   3   6
>>> df['c1']
i1    1
i2    2
i3    3
Name: c1, dtype: int64
>>> df.ix['i1']
c1    1
c2    4
Name: i1, dtype: int64
>>> df.loc['i1','c1']
1
```

- **augment data** (e.g. merge, append, etc.) in any way to consolidate data from various sources

```
>>> df1=DataFrame({'one':Series([1,2,3],index=
['a','b','c']),'two':Series([6,7,8],index=['a','b','c'])})
>>> df1
  one  two
a   1    6
b   2    7
c   3    8
>>> df2=DataFrame({'one':Series([4,5],index=['e','f']),'two':Series([9,10],index=
['e','f'])})
>>> df2
  one  two
e   4    9
f   5   10
>>> df1.append(df2)
  one  two
```

```
a  1  6
b  2  7
c  3  8
e  4  9
f  5  10

>>> df3=DataFrame({'three':Series([11,12,13],index=
['a','b','c']),'four':Series([16,17,18],index=['a','b','c'])})
>>> df3
  four  three
a  16    11
b  17    12
c  18    13
>>> merge(df1,df3,left_index=True, right_index=True)
  one  two  four  three
a  1   6   16    11
b  2   7   17    12
c  3   8   18    13
```

- **clean data** in any way (e.g. remove, replace, repair, resample, etc.)

```
>>> s=Series([1,2,-3,4,5],index=date_range('2014-1-1',periods=5,freq='30Min'))
>>> s
2014-01-01 00:00:00    1
2014-01-01 00:30:00    2
2014-01-01 01:00:00   -3
2014-01-01 01:30:00    4
2014-01-01 02:00:00    5
Freq: 30T, dtype: int64
>>> s[s<0]=nan
>>> s
2014-01-01 00:00:00    1
2014-01-01 00:30:00    2
2014-01-01 01:00:00   NaN
2014-01-01 01:30:00    4
2014-01-01 02:00:00    5
Freq: 30T, dtype: float64
>>> s1=s.resample('H',fill_method='pad')
>>> s1
2014-01-01 00:00:00    1.5
2014-01-01 01:00:00    4.0
2014-01-01 02:00:00    5.0
Freq: H, dtype: float64
>>> s1.resample('30Min',fill_method='pad')
```

```
2014-01-01 00:00:00    1.5
2014-01-01 00:30:00    1.5
2014-01-01 01:00:00    4.0
2014-01-01 01:30:00    4.0
2014-01-01 02:00:00    5.0
Freq: 30T, dtype: float64
```

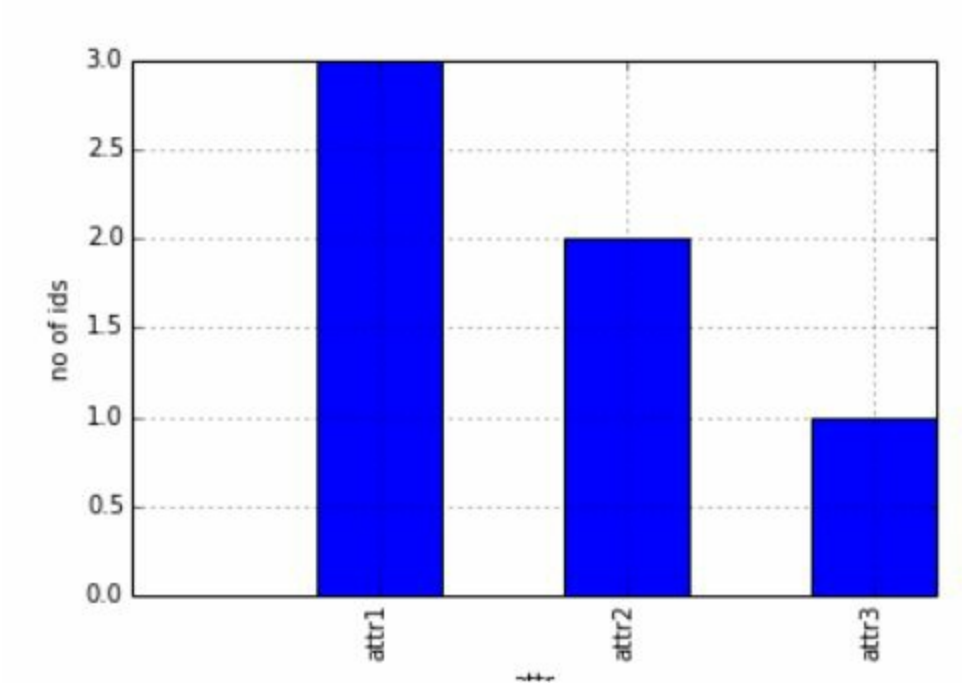
- **slice data** in any way (e.g. aggregate over rows/records, correlate over columns/variables, etc.) to extract information from data

```
>>> df=DataFrame({'one':Series([1,2,3,4,5],index=['a','b','c','d','e']),...
'two':Series([6,7,8,9,10],index=['a','b','c','d','e'])})
>>> df
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
>>> df.describe()
             one        two
count  5.000000   5.000000
mean   3.000000   8.000000
std    1.581139   1.581139
min    1.000000   6.000000
25%    2.000000   7.000000
50%    3.000000   8.000000
75%    4.000000   9.000000
max    5.000000  10.000000
```

By "group by" we are referring to a process involving one or more of the following steps

- Splitting the data into groups based on some criteria
- Applying a function to each group independently
- Combining the results into a data structure

```
from pylab import*
from pandas import *

df=DataFrame([['name1','attr1'],['name2','attr1'],['name3','attr1'],
              ['name4','attr2'],['name5','attr2'],['name6','attr3']],
             index=['id1','id2','id3','id4','id5','id6'],
             columns=['name','attr'])
grouped=df.groupby(['attr'],sort=True)['attr']
grouped.count().plot(kind='bar'); ylabel('no of ids')
```



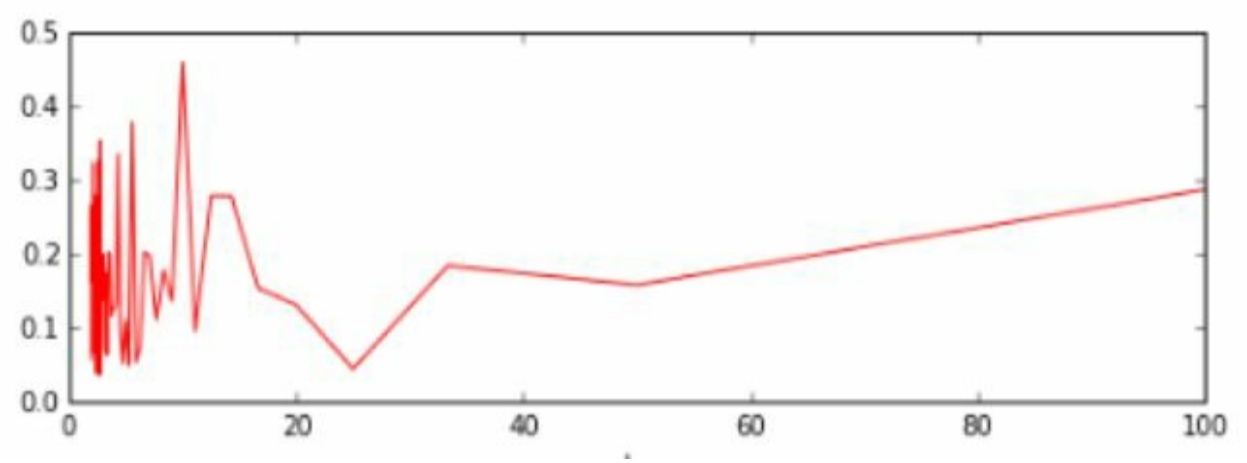
Analytics

Since most data is time series, here we use 2 simple methods to study anomalies in high, medium and low frequency.

Single Variable

Frequency domain

```
figure(figsize=(8,2.5));
fft_output = rfft(np.random.randn(100))
magnitude_only = [sqrt(i.real**2 + i.imag**2)/len(fft_output) for i in fft_output]
hours = [1.0/((i*1.0/100)*1.0) for i in range(1,100/2+1)]
plot(hours, magnitude_only[1:], 'r'); xlim([0,100]); xlabel('hr');
```



- Low time interval (i.e. high frequency) may indicate noise/control chattering
- Middle time interval (~24hr/1day) daily operation variations
- Long time interval (>48hr/2days) plant upset/shutdown

Time domain

Time domain analysis can reveal the same information. We can identify the moving trends and dynamic trend w.r.t. analysis windows of different sizes.

```
s1=np.random.randn(100)
#24 hrs mean prediction
N=48;
sn_mean=list(rolling_mean(s1,N));
sn_std=list(rolling_std(s1,N));
b=sn_mean[-N:];
t=[i for i in range(N)];
```

```
A=vstack([t, ones_like(t)]).T
xn=linalg.lstsq(A, b)[0];
tn_mean=[xn[0]*(N+ti)+xn[1] for ti in range(N/2)]
```

#**3 hr mean prediction**

```
M=6;
sm_mean=list(rolling_mean(s1,M));
sm_std=list(=rolling_std(s1,M));
b=sm_mean[-M:];
t=[i for i in range(M)]; A=vstack([t, ones_like(t)]).T
xm=linalg.lstsq(A, b)[0];
tm_mean=[xm[0]*(M+ti)+xm[1] for ti in range(M/2)]
```

#**3 hr dynamic prediction**

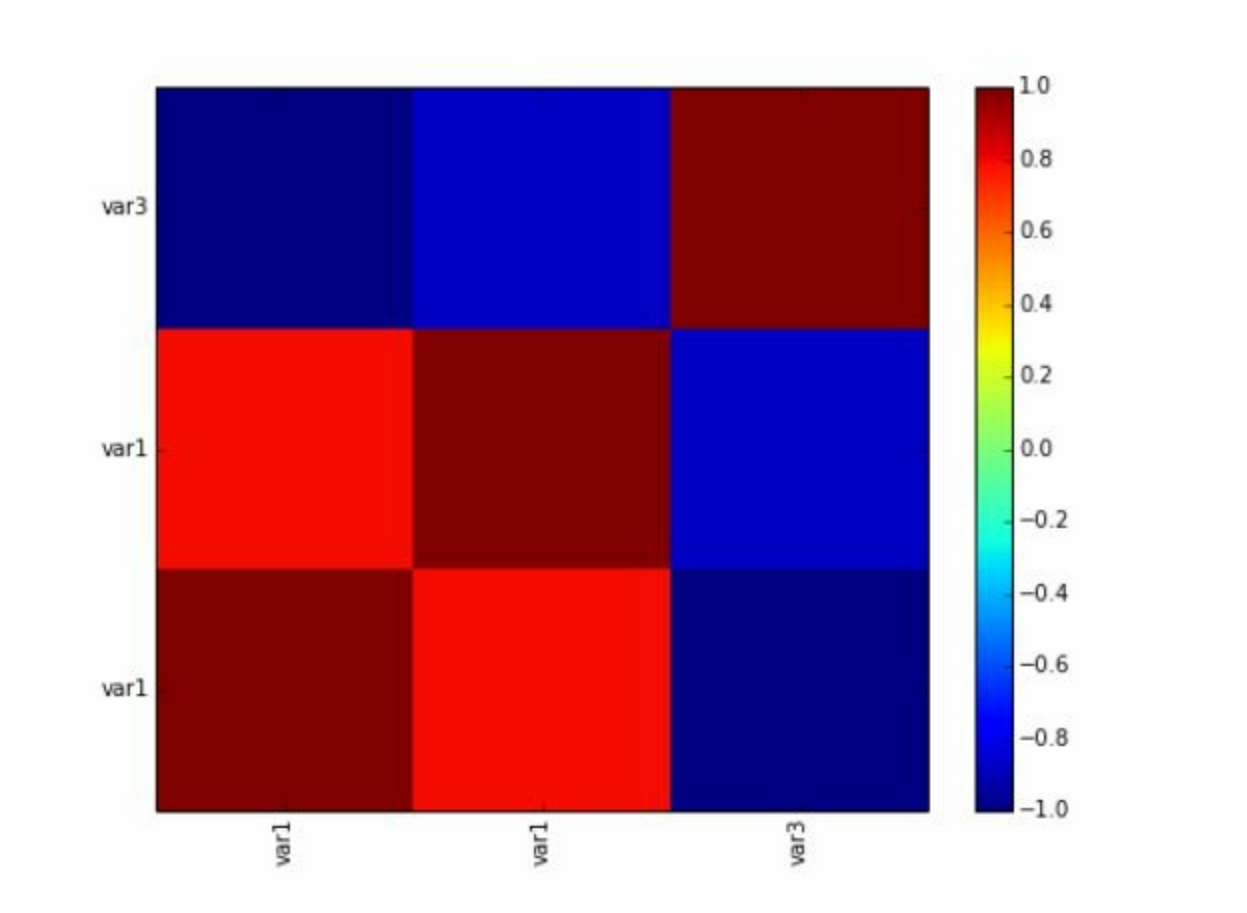
```
b=sm_mean[-M:]; A=zeros((M,2),float);
for i in range(M):
  A[i,]=sm_mean[-3-i:-1-i]
xd=linalg.lstsq(A,b[::-1])[0];
dm=zeros(M/2);
for i in range(M/2):
   dm[i]=dot(b[-2:],xd); b=b[-1:]+[dm[i]]
```

Based on the predictions of different trends, we can derive

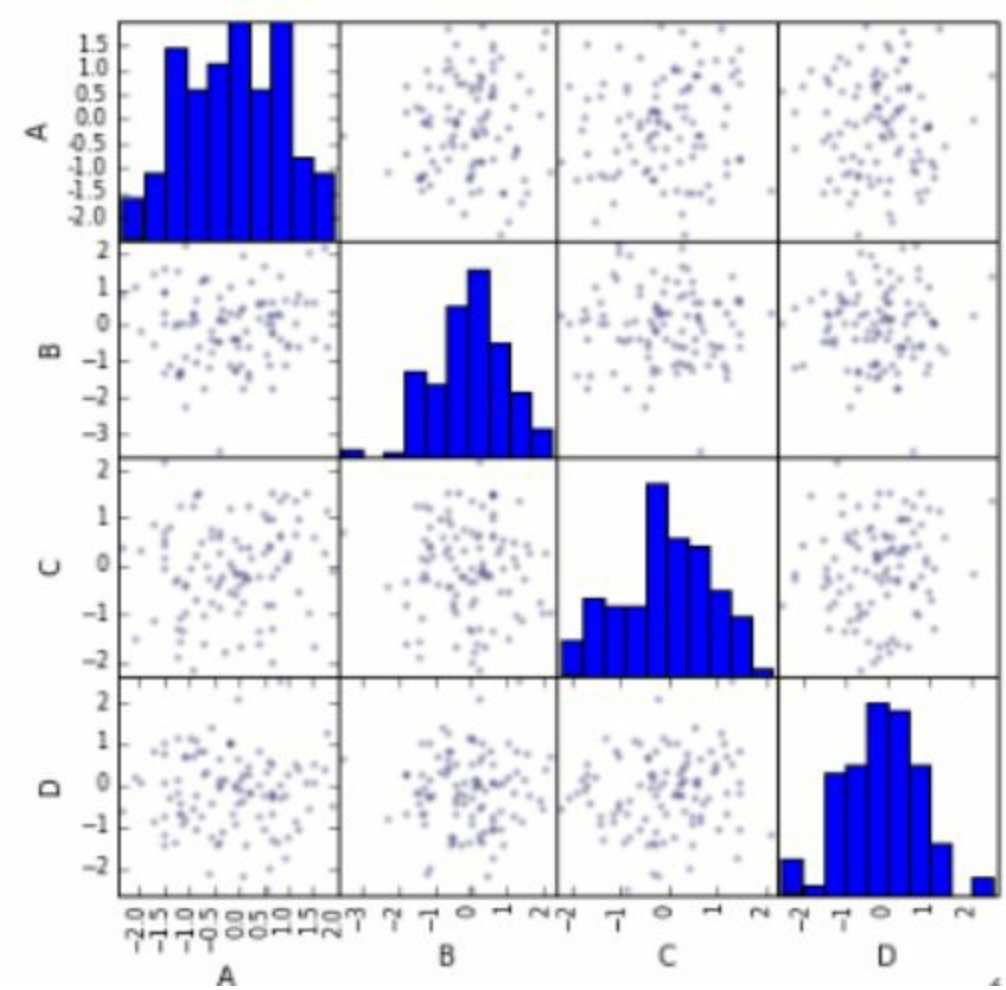
- Short-term std and dynamic rate of change are high, indicating dramatic system upset
- High long-term and short-term rates of change indicate lack of control
- Low long-term and short-term rates of change but high short-term std indicates noise/control chattering

Multi-Variables
Correlation

```
from pylab import *
from pandas import *

df=DataFrame([[1,5,6.0],[2,4.8,5],[3,6.2,3.2]],index=['id1','id2','id3'],columns=
['var1','var1','var3'])
R = corrcoef(df.T)
figure(figsize=(8,6))
plt=pcolor(R)
colorbar()
clim(-1,1)
yticks(arange(0.5,len(df.columns)+0.5),df.columns)
xticks(arange(0.5,len(df.columns)+0.5),df.columns, rotation=90);
```



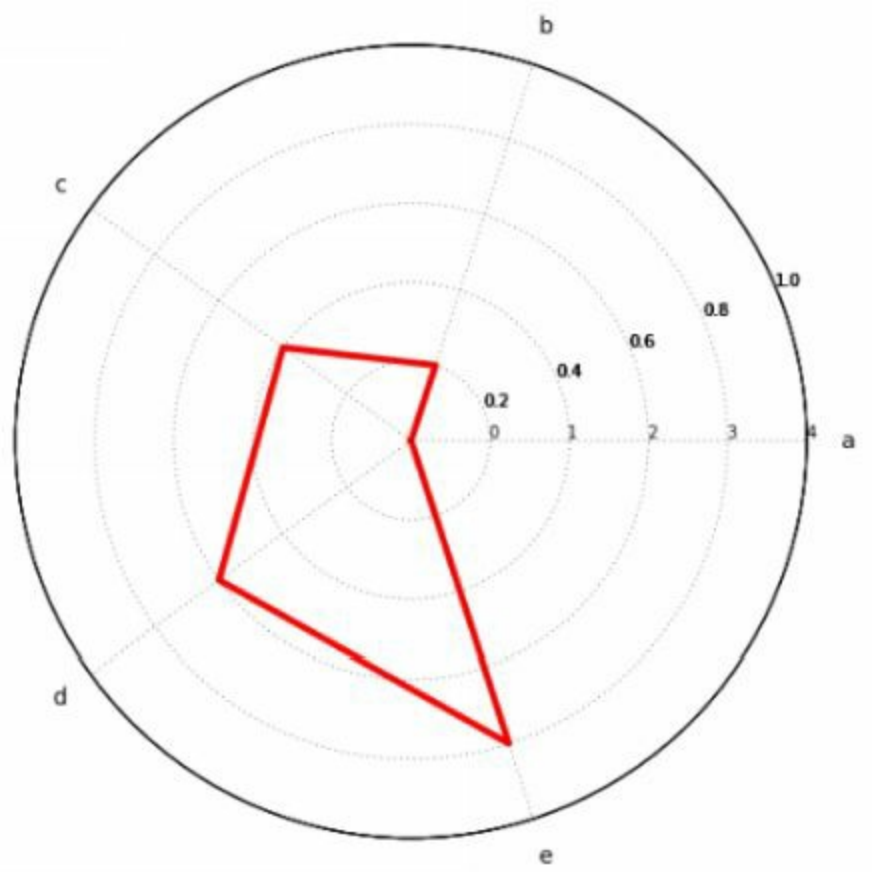

```
df=DataFrame(np.random.randn(100, 4), columns=list('ABCD'))
figure(); scatter_matrix(df, alpha=0.2, figsize=(6,6));
```


A
B
C
D

Radar chart

```
class Radar(object):
    def __init__(self, fig, titles, labels, rect=None):
        if rect is None: rect = [0.05, 0.05, 0.95, 0.95]

        self.n = len(titles)
        self.angles = arange(0, 0+360, 360.0/self.n)
        self.axes = [fig.add_axes(rect, projection="polar", label="axes%d" % i)
                     for i in range(self.n)]

        self.ax = self.axes[0]
        self.ax.set_thetagrids(self.angles, labels=titles, fontsize=14)

        for ax in self.axes[1:]:
            ax.patch.set_visible(False)
            ax.grid("off")
            ax.xaxis.set_visible(False)

        for ax, angle, label in zip(self.axes, self.angles, labels):
            ax.set_rgrids(range(1, 6), angle=angle, labels=label)
            ax.spines["polar"].set_visible(False)
           ax.set_ylim(0, 5)

    def plot(self, values, *args, **kw):
        angle = deg2rad(r_[self.angles, self.angles[0]])
        values = r_[values, values[0]]
        self.ax.plot(angle, values, *args, **kw)
```

```
#generate radar chart
fig = plt.figure(figsize=(7,7))
titles = 'radar'
labels = [[0,1,2,3,4]*5]
radar = Radar(fig, titles, labels)
line = [0,1,2,3,4]
radar.plot(line,"-",lw=4,color='red')
```


b
c
a
d
e
0.2
0.4
0.6
0.8
1.0
0
1
2
3
4

Report /Email/Survey

The media for our analytics product to reach users is through email, sending report and surveying customers to get “soft” feedback.

Report.

Report can be written in HTML, then printed in Microsoft Words as pdf

```
from win32com.client import *
f = open('Report.html','w')
message = """
        <!DOCTYPE html>
        <html>
        <head>
        </head>
        <body>
        </body>
        </html>"""
f.write(message)
f.close()

word = Dispatch('Word.Application')
doc = word.Documents.Add('Report.html')
doc.SaveAs('Report.pdf', FileFormat=17)
doc.Close()
word.Quit()
```

Survey Email.

Survey can be written in HTML in Email and sent with attached pdf report.

```
from win32com.client import *
o = Dispatch("Outlook.Application")
Msg = o.CreateItem(0)
Msg.Subject = "Report and Survey"
Msg.HTMLBody =  """<!DOCTYPE html>
        <html>
        <body>
          <p>Please reply this survey by answering the following questions</p>
           <b>Q1: Was the variation caused by</b><br>
           <input type="radio" name="Cause" value="Bad Probe">Bad Probe<br>
           <input type="radio" name="Cause" value="Control Issue">Control Issue<br>
```

```
            <input type="radio" name="Cause" value="System Upset">System Upset<br><br>
        </body>
        </html>"""
Msg.Attachments.Add("Report.pdf")
Msg.Send()
```

Simulation and Python Class

The simplicity of defining a class makes it easy to build a simulation in Python and programmatically achieve the similar flow as in Matlab Simulink.

For example, for a continuous stirring tank reactor (CSTR), the following shows how to apply simple on/off to control OUT flow in order to maintain OUT concentration, given variations in IN flow and IN concentration.

```
# simulation
plant = Plant(V, C)
ctrl_OUT = ON_OFF_Ctrl(OUTmin, OUTmax, Cmin, Cmax)
closedLoop = CL_Ctrl(plant,ctrl_OUT)
simulation = array([closedLoop.ctrl(IN, IN_C) for i in range(0,50)])

class Plant:
    def __init__(self, _V, _C):
        self.V = _V
        self.C = _C
    def OneStep(self, _IN, _IN_C, _OUT):
        self.C = (1-_OUT/self.V)*self.C + _IN/self.V*_IN_C
        return self.C

class ON_OFF_Ctrl:
    def __init__(self, _OUTmin, _OUTmax, _Cmin, _Cmax):
        self.OUTmin = _OUTmin
        self.OUTmax = _OUTmax
        self.Cmin = _Cmin
        self.Cmax = _Cmax
        self.C = 0
        self.cv = 0
    def CV(self, _C):
        self.C = _C
        if self.C > self.Cmax: self.cv = self.OUTmin
        if self.C < self.Cmin: self.cv = self.OUTmax
        return self.cv

class CL_Ctrl:
    def __init__(self, _Plant, _Ctrl_BD, _Ctrl_F):
        self.Plant = _Plant
        self.Ctrl_OUT = _Ctrl_OUT
        self.C = 0;
        self.cv_OUT = 0;
```

```
def ctrl(self, _IN, _C):
    self.C = self.plant.OneStep(_IN, _IN_C, self.cv_OUT)
    self.cv_OUT = self.Ctrl_OUT.CV(self.C)
   return self.C
```

Deployment to Cloud

Install command line client for Cloud foundry https://docs.cloudfoundry.org/cf-cli/install-go-cli.html

In your command window (here is the cf command cheatsheet http://www.oneguyinit.com/wp-content/uploads/2014/04/CF-Cheat-Sheet.png)

- cf login
- API endpoint e.g. pivotal, bluemix, amazon, etc.
- Push the files along with
    - **manifest.yml** to specify commands for cloud deployment, and
    - **environment.yml** to specify required software environment in cloud

Note the difference between local deployment and cloud deployment is that the Cloud will specify port and host (refer to https://github.com/ihuston/python-cf-examples for example)

```
port = int(os.getenv("VCAP_APP_PORT"))
app = SimpleSineApp()
app.launch(port=port, host='0.0.0.0')
```

Collaboration on Github or Ipython Notebook and FTP Server

Github

Setup github account and synch local code with github account (https://help.github.com/articles/set-up-git/)

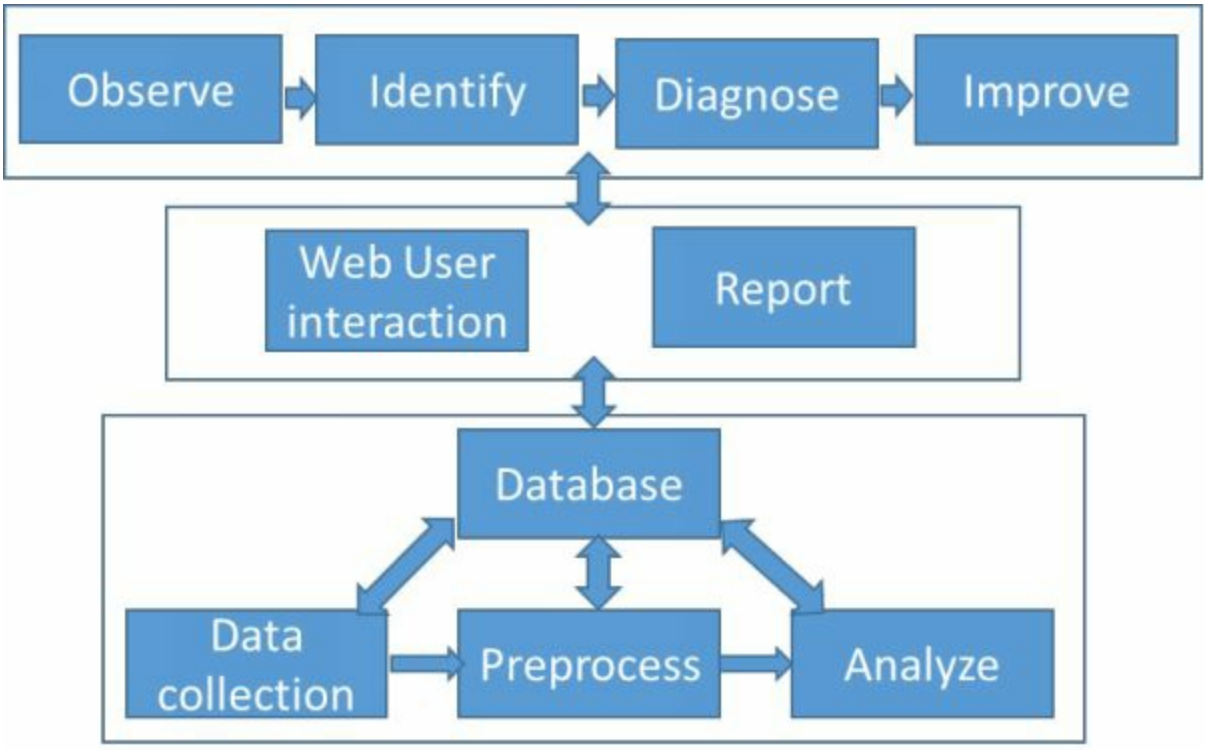
```
eval $(ssh-agent -s)
ssh-add ~/.ssh/####
git clone ####
git status
git add .
git status
git commit -m "description here"
git status #(local saved)
git push #(push to github) or git push origin master
git tag v1.0 -m "description here" #save version local
git push --follow-tags #save to server
```

ipython and ftp servers

- setup remote access of network ipython notebook server in a folder at a computer (https://ipython.org/ipython-doc/1/interactive/public_server.html)
- setup fpt server to upload and download files for ipython notebook server in the folder

An Observe-Identify-Diagnose-Improve Analytics Product

The architecture

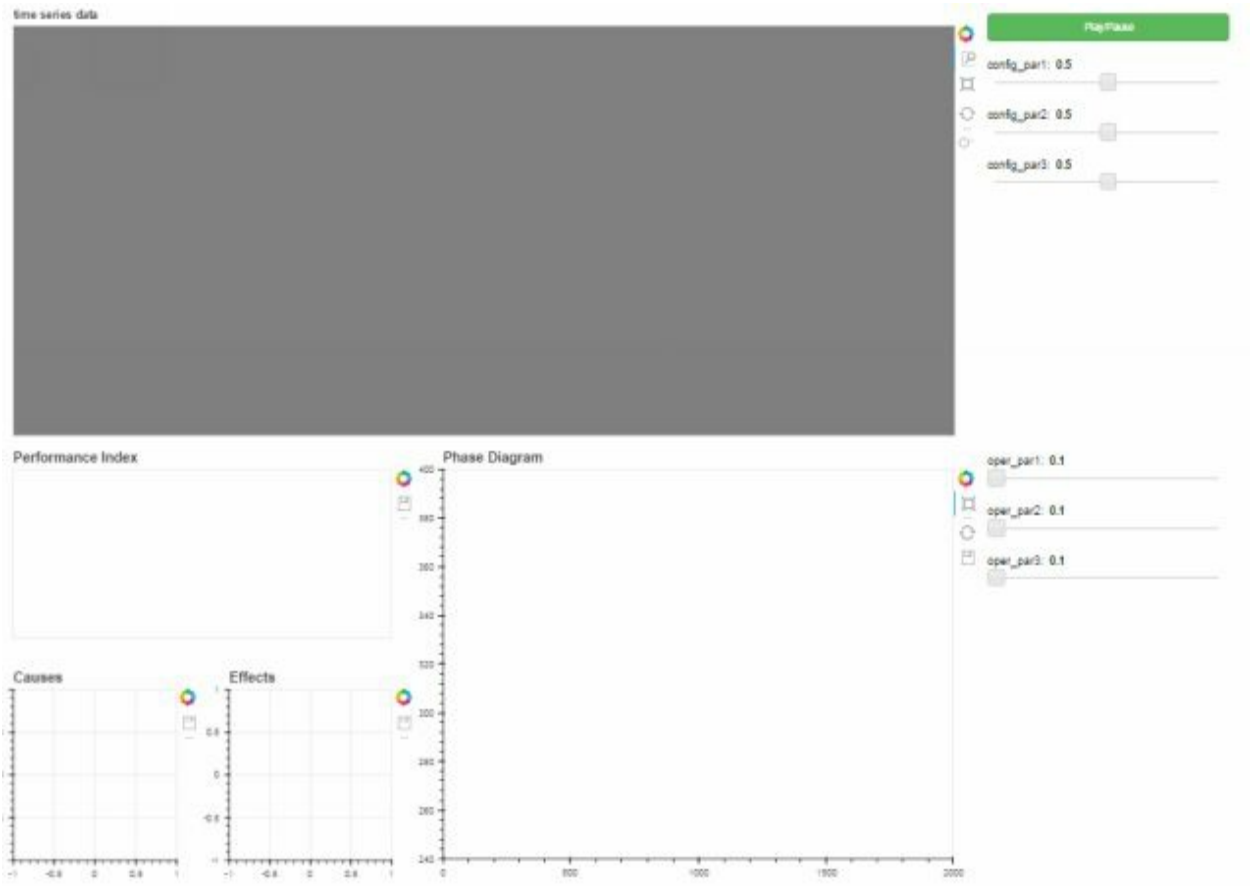


Comparing to the first analytics product, this product allows a user to go through a cognitive process to observe, identify, diagnose the problems and improve the system performance.

Web User Interactive Interface

Bokeh (http://bokeh.pydata.org/en/latest/) is a Python interactive visualization library that targets modern web browsers for presentation. Its goal is to provide elegant, concise construction of novel graphics in the style of D3.js, and to extend this capability with high-performance interactivity over very large or streaming datasets. Bokeh can help anyone who would like to quickly and easily create interactive plots, dashboards, and data applications.

In its core, it has 3 parts (see the example user interface)



- define widgets and layouts
- define triggers and their corresponding actions
- push the document to bokeh server

```
from bokeh.layouts import column, row
from bokeh.models.widgets import TextInput, Toggle
from bokeh.io import curdoc

comment=TextInput(value="", title="Comment:");
```

```
submit_toggle = Toggle(label="submit a comment", type="success")

def submit_handler(active):
   #save comment

submit_toggle.on_click(submit_handler)

vbox=column([comment,submit_toggle])
curdoc().add_root(vbox)
```

Bokeh makes seamless integration of front end (client side) web interface and back end (server side) python ecosystem. Python defined web interface widgets, layouts and interactions are seamlessly translated into JavaScript so that developers don’t need to learn to write JavaScript.

Observe and Identify – Interactive play (periodic callback)

```
def play_pause_toggle_handler(active):
   if active:
      curdoc().add_periodic_callback(Play,1000) #check every sec
      status.value='play'
   else:
      curdoc().remove_periodic_callback(Play)
      status.value='pause'
```

Diagnose and improve – UI widget Series or DataFrame

Using Pandas Series and DataFrame data structure is very effective in defining a set of input UIs for diagnosis and improvement activities.

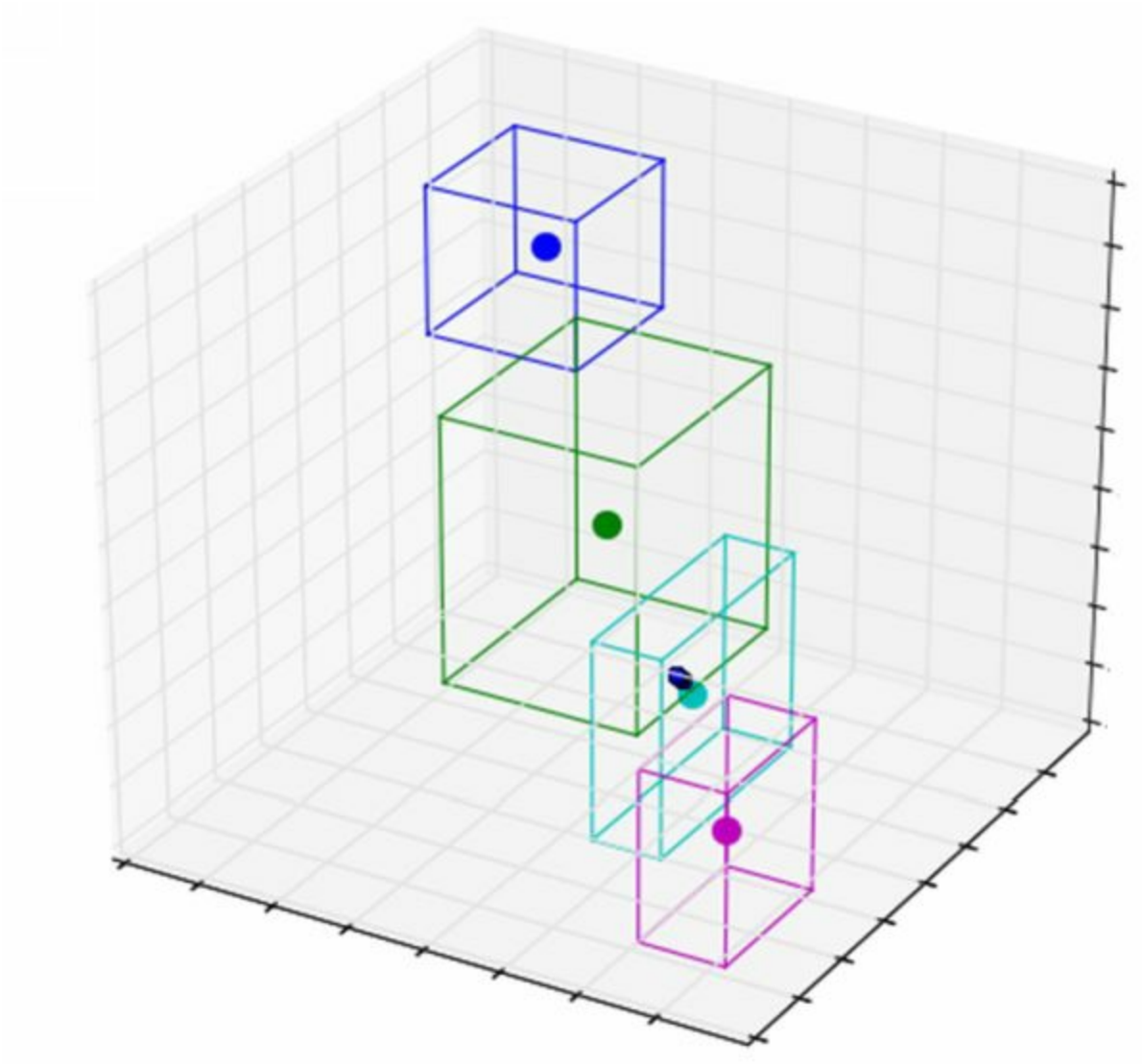
```
from pandas import *
from bokeh.models.widgets import sliders

sliders=Series([Slider(title=var) for var in ['a','b','c','d','e']])

def input_change(attrname, old, new):
   #action for input_change

for var in sliders.index:
   sliders[var].on_change('value', input_change)
```

Report with 3D and Animation

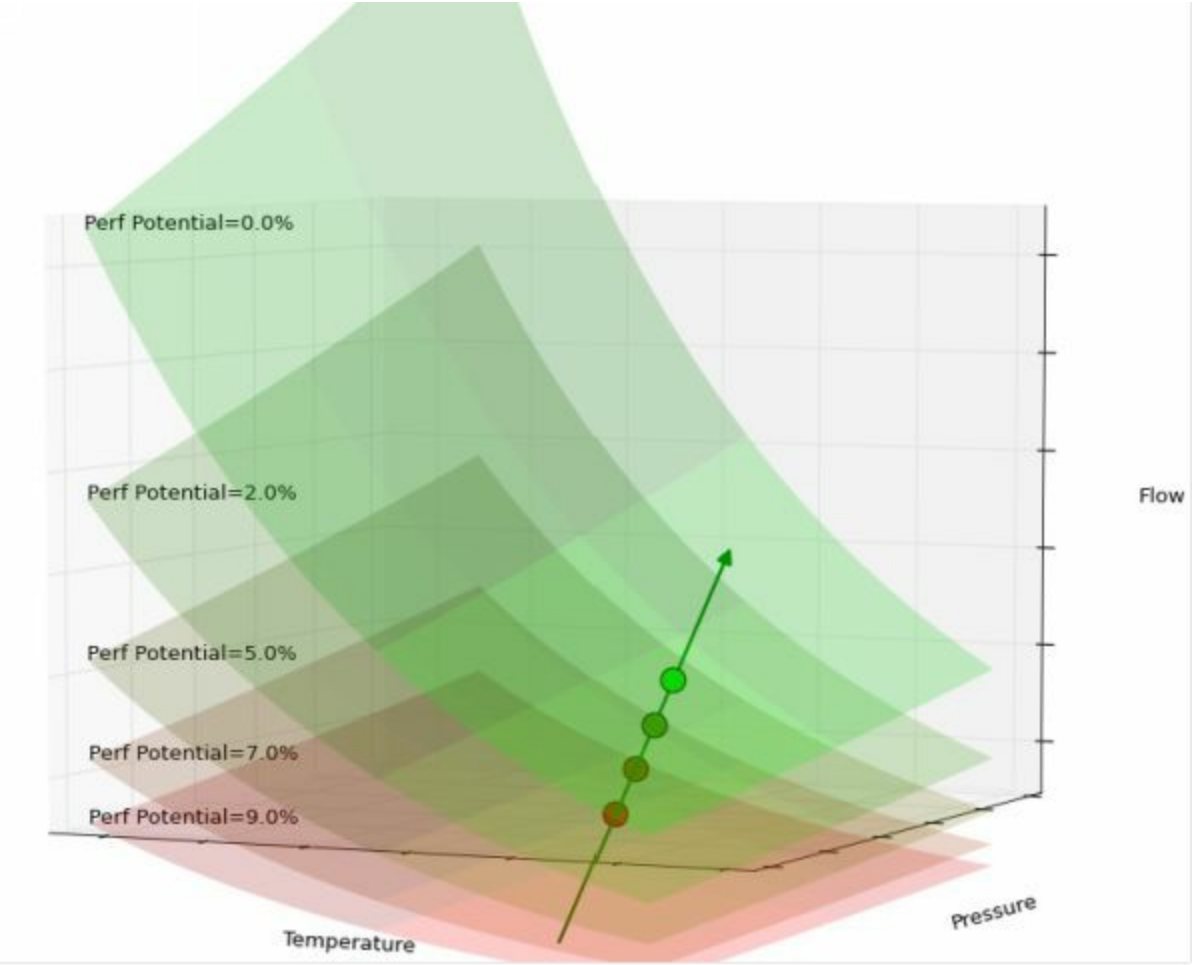
Python package Matplotlib (http://matplotlib.org/) provides 3D plot and animation capabilities, that can further enhance the report content.

3D graph

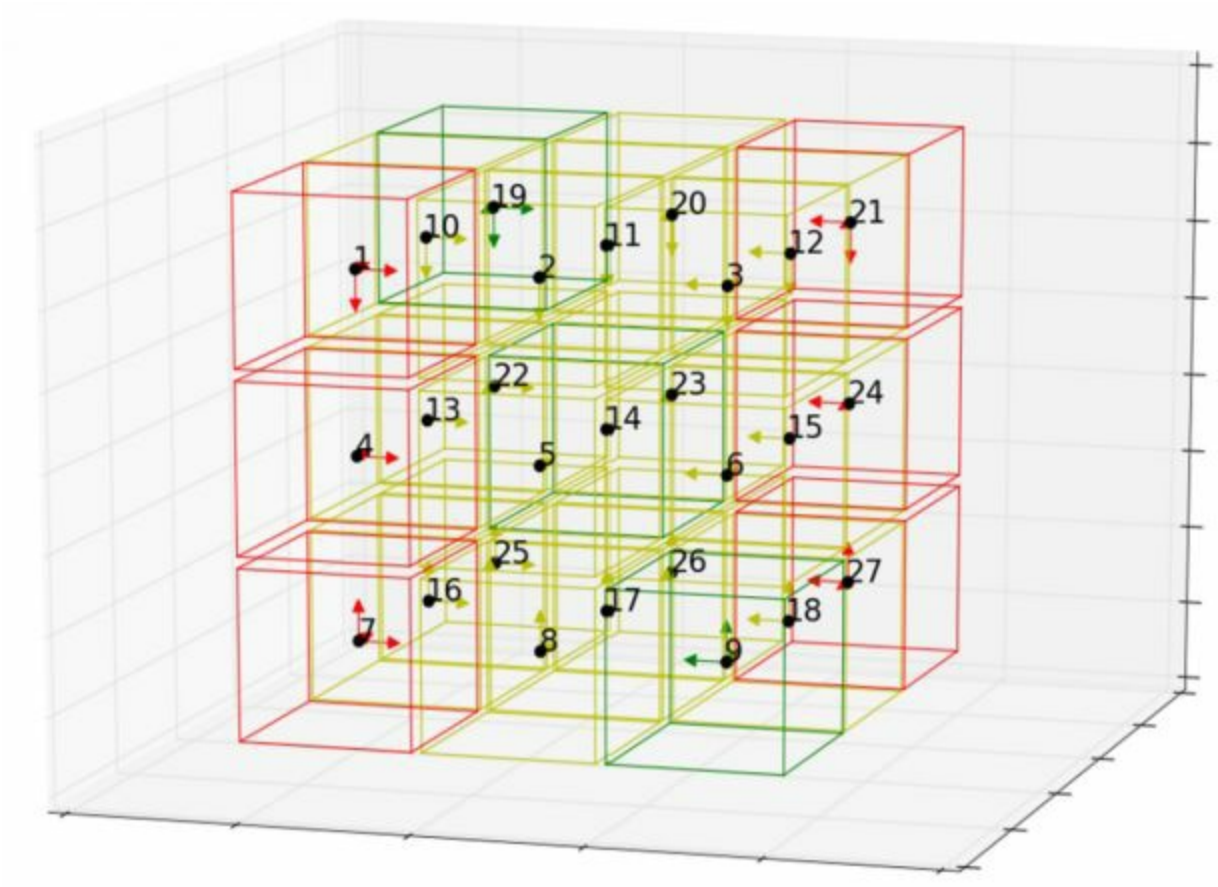
```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from itertools import product, combinations
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

def draw_box(_X, _Y, _Z, _color):
    for s, e in combinations(array(list(product(_X,_Y,_Z))), 2):
        if (sum(abs(s-e)) == abs(_X[1] - _X[0])) or \
           (sum(abs(s-e)) == abs(_Y[1] - _Y[0])) or \
           (sum(abs(s-e)) == abs(_Z[1] - _Z[0])) :
            ax.plot3D(*zip(s,e), color=_color, ls=':')
draw_box(X,Y,Z, 'c')
```



You can use 3D plot to generate iso-surfaces, or



partition the 3D state space into finite elements


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27

3D animation

```
import matplotlib.pyplot as plt
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.animation as animation

fig = plt.figure()
# Set up formatting for the movie files
Writer = animation.writers['ffmpeg']
writer = Writer(fps=5, metadata=dict(artist='Me'), bitrate=1800)

def update_line(i, line) :
   line.set_data(data[0:2, i:i+1])
   line.set_3d_properties(data[2, i:i+1])
   return line,

line_ani = animation.FuncAnimation(fig, update_line, np.arange(1,199),\
                        fargs=(line), interval=50, blit=True)
line_ani.save('###.mp4', writer=writer)
```

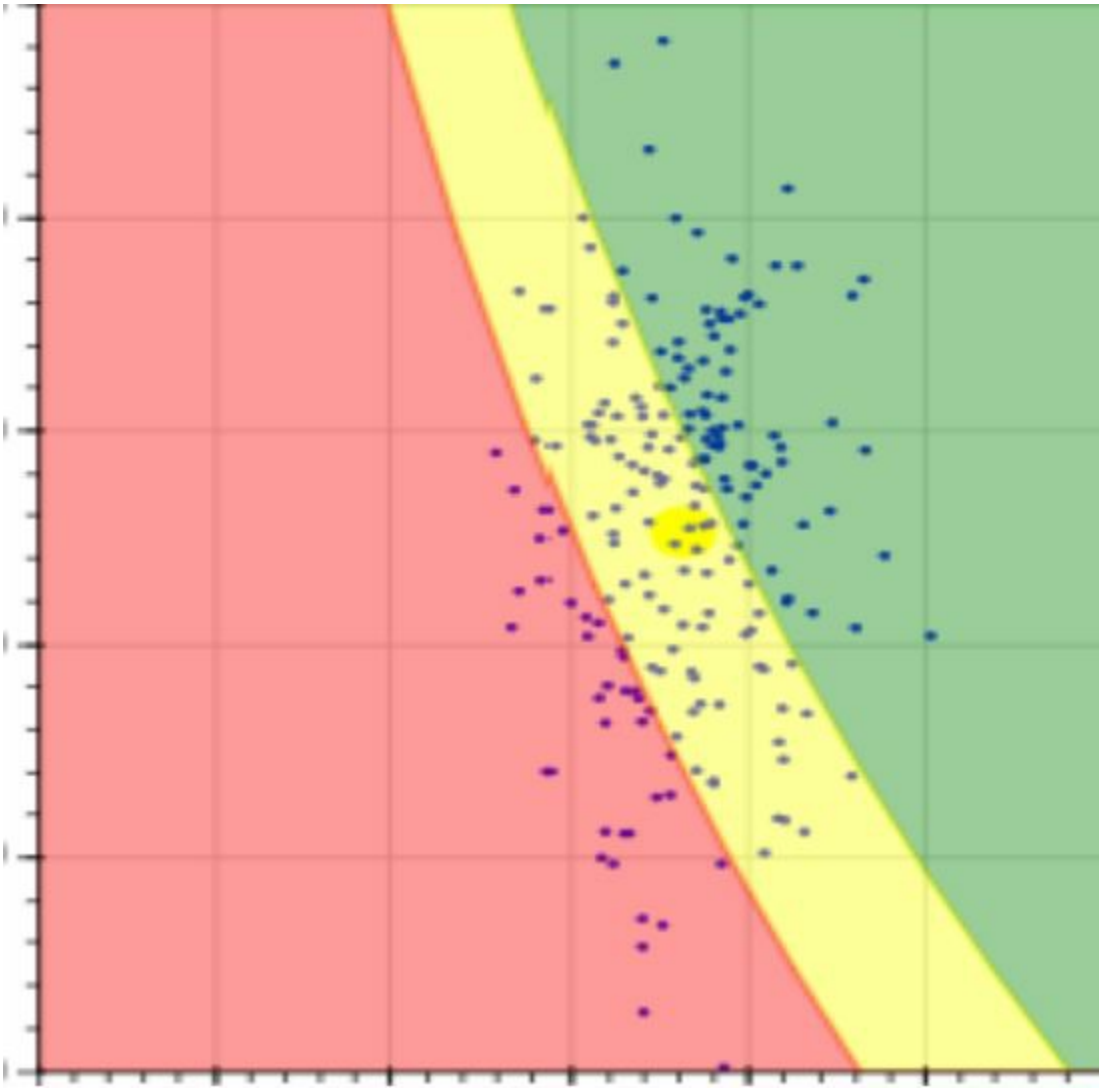
In “animation.FuncAnimation”, the first parameter is the figure, then the “update_line” event handler, followed by the specifications of the parameters for the event handler “update_line”. The generated mp4 file can be attached with the report via email.

Analyze, Quantify Risk and Tradeoff Return and Risk

The analytics in the first product is for basic asset performance. The next level of analytics is to perform risk reward analytics.

Quantify the risk by one-line Monte Carlo simulation

Suppose the state space is separated into safe region in green, risk region in red, and uncertain region in yellow. The risk probability of getting into red region can be quantified by one-line Monte Carlo simulation as follows



MonteCarlo=\

len(filter((lambda i: function(x[i],y[i])>boundary[i]), range(100)))*1.0/100

Where x,y,boundary are random variables of certain probability distributions, representing x and y axes and the yellow boundary.

Tradeoff between Risk and Return

The tradeoff between risk and return can be formulated as a convex optimization problem (https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf), for which we can use Python packages cvxopt and easy-to-use API cvxpy (explained in later section about "convex optimization").

$$\text{minimize } -\bar{p}x + ax^T\Sigma x$$

$$\text{subject to } 1^Tx = 1, x \geq 0$$

where

x denotes the set of assets held,

$\bar{p}$ is relative mean price change of the set of asset and $\bar{p}x$ is the mean return,

Σ is covariance and $x^T\Sigma x$ is the mean return variation, i.e. the risk,

a is the tradeoff between return and risk, $a = 0$ corresponds to maximum return, while a approaching ∞ represents minimum risk.

$1^Tx = 1$ is budget constraint

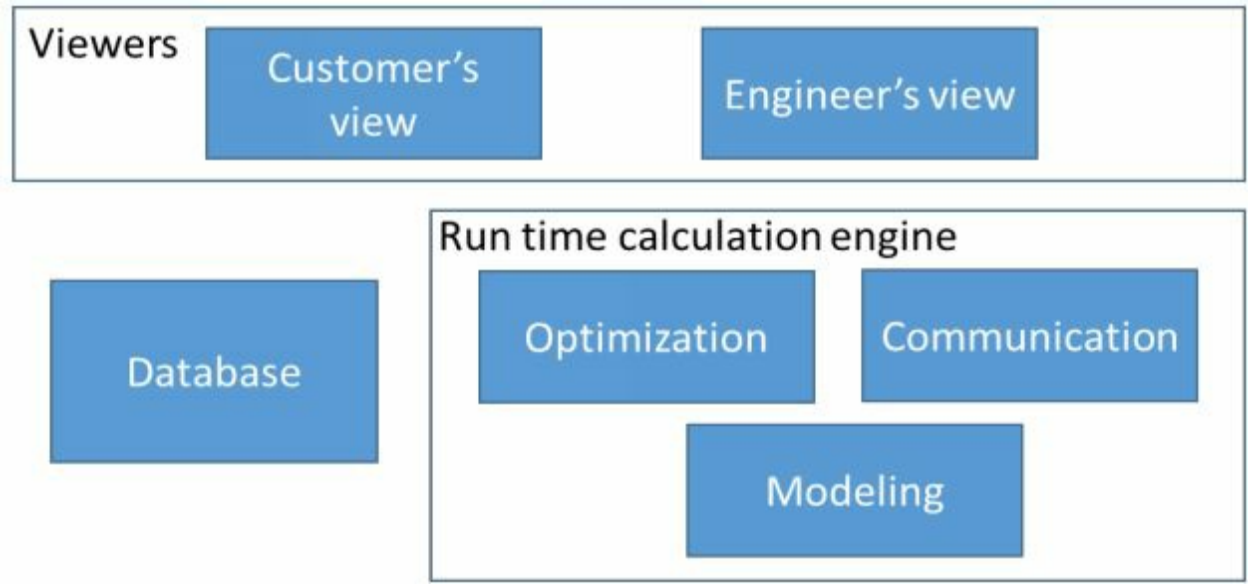
Modular Approach

As more contents are added into analytics, it is important to separate contents into different modules so that maximum independency can be achieved, thus minimum interference between updates for different contents.

As shown in the architecture, database in the center is for sharing, it is the only place the contents intercept, thus complete separation of modules is achieved by only interfacing them with database.

A Supervisory Modeling, Optimization and Learning Product

The Architecture



This product is different in that the product is no longer servicing customer remotely, it is located at customer site. The closeness to customer makes it possible for more comprehensive data communication, more detailed modeling and more real time decision making optimization and control.

User Interface

Qt (https://www.qt.io/developers/) is the primary tool for UI design. PyQt (https://wiki.python.org/moin/PyQt) provides a python front end to use Qt.

The basic flow of using Qt to develop UI is the same as that of using bokeh, i.e.

- define widgets and layouts
- define triggers and their corresponding actions
- instead, package into executable and distribute it to customer site

We can use the graphic design tool qt-designer to drag and drop widgets and design layouts (i.e. WYSIWYG, what you see is what you get)

```
import sys

from PyQt4 import QtGui, QtCore

class Example(QtGui.QWidget):

    def __init__(self):
        super(Example, self).__init__()
        self.simulationPushButton = QtGui.QPushButton('simulationPushButton', self)
        QtCore.QObject.connect(self.simPushButton, \
                               QtCore.SIGNAL("clicked()"), self.simulation)

    def start(self):
        self.timer=self.startTimer(10)

    def timerEvent(self, evt):
        #do timerEvent
        if self.cnt == 99:
            self.killTimer(self.timer)
            self.simulationPushButton.setEnabled(True)
        else:
            self.cnt +=1
            self.simulationPushButton.setEnabled(False)

    def simulation(self):
        #do simulation
        self.cnt = 0
        self.timerEvent(None)
        self.start()

def main():
    app = QtGui.QApplication(sys.argv)
    ex = Example()
```

```
    ex.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

Communications

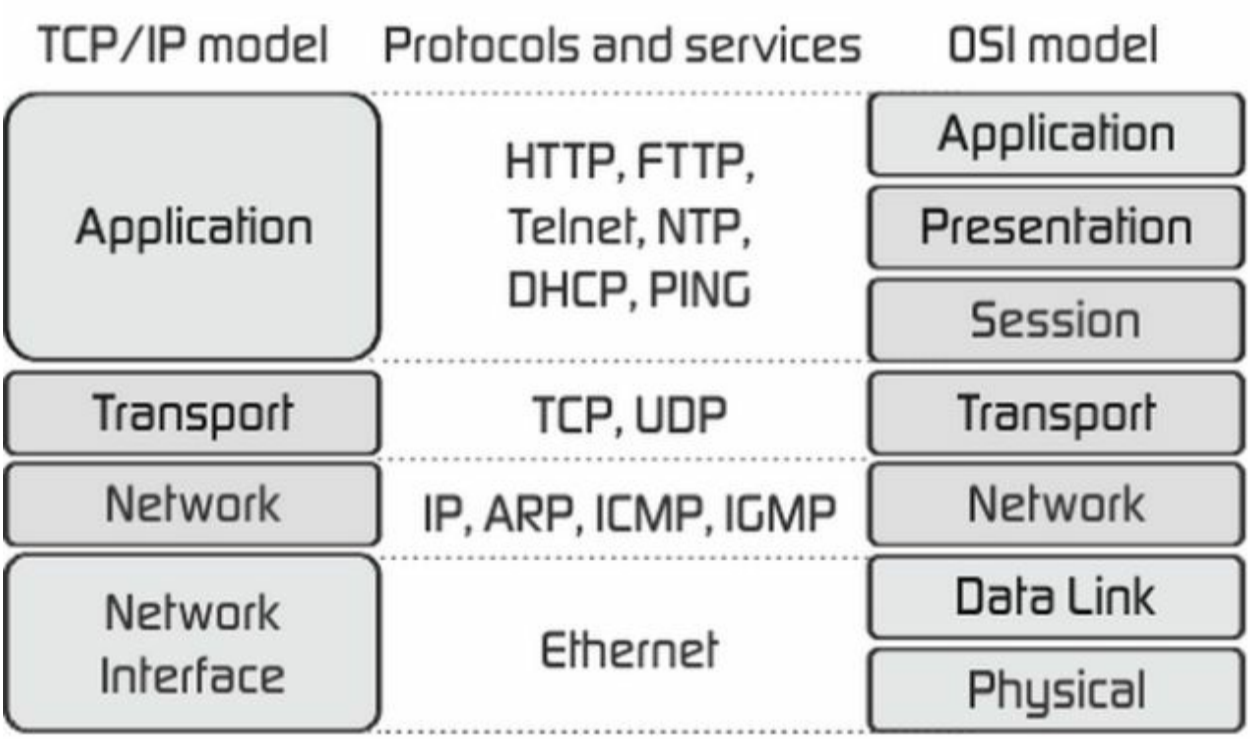
Serial and Parallel interfaces

Out of the box, Python cannot directly access the underlying hardware, nor can it interface directly with the software library modules provided by most hardware vendors. (Yet, there are open source hardware platforms, such as raspberry Pi, beagle bone, Arduino, where Python can be used via GPIB to access hardware.)

One way to give Python access to the real world is to use the C programming language to create extensions. These are basically dynamically linked library (DLL) objects that give Python capabilities beyond what is included in its own libraries, mainly using extensions as “wrappers” for the DLL modules supplied with commercial instruments and interfaces.

Another approach is to use Python’s ctypes library, which contains methods to directly access the functions in an external DLL. The ctypes library can be used directly from within a Python program without the need to write any C code.

The most commonly encountered types of serial interfaces, are RS-232, RS-485 and USB. And GPIB/IEEE-488 is parallel interface. All communication interfaces are based on the TCP/IP model or OSI model


TCP/IP model
Protocols and services
OSI model
Application
HTTP, FTTP,
Telnet, NTP,
DHCP, PING
Application
Presentation
Session
Transport
TCP, UDP
Transport
Network
IP, ARP, ICMP, IGMP
Network
Network
Interface
Ethernet
Data Link
Physical

OPC Server and Client

Python OPCUA (https://github.com/FreeOpcUa) supports two protocols. This is visible to application programmers only via changes to the URL. The binary protocol is **opc.tcp://Server** and **http://Server** is for Web Service.

Start server

```
import sys
sys.path.insert(0, "..")
import time
from opcua import ua, Server

if __name__ == "__main__":

    # setup our server
    server = Server()
    server.set_endpoint("opc.tcp://0.0.0.0:4840/freeopcua/server/")

    # setup our own namespace, not really necessary but should as spec
    uri = "http://examples.freeopcua.github.io"
    idx = server.register_namespace(uri)

    # get Objects node, this is where we should put our nodes
    objects = server.get_objects_node()

    # populating our address space
    myobj = objects.add_object(idx, "MyObject")
    myvar = myobj.add_variable(idx, "MyVariable", 6.7)
    myvar.set_writable()    # Set MyVariable to be writable by clients

    # starting!
    server.start()

    try:
        count = 0
        while True:
            time.sleep(1)
            count += 0.1
            myvar.set_value(count)
    finally:
```

```
#close connection, remove subcsriptions, etc
server.stop()
```

Client access

```
import sys
sys.path.insert(0, "..")
from opcua import Client

if __name__ == "__main__":

    client = Client("opc.tcp://localhost:4840/freeopcua/server/")
    # client = Client("opc.tcp://admin@localhost:4840/freeopcua/server/") #connect using a
user
    try:
        client.connect()

        # Client has a few methods to get proxy to UA nodes that should always be in address
space such as Root or Objects
        root = client.get_root_node()
        print("Objects node is: ", root)

        # Node objects have methods to read and write node attributes as well as browse or
populate address space
        print("Children of root are: ", root.get_children())

        # Now getting a variable node using its browse path
        myvar = root.get_child(["0:Objects", "2:MyObject", "2:MyVariable"])
        obj = root.get_child(["0:Objects", "2:MyObject"])
        print("myvar is: ", myvar.get_value())

    finally:
        client.disconnect()
```

Modeling

In mathematics, statistics, and computational modelling, a grey box model combines a partial theoretical structure with data to complete the model. The theoretical structure may vary from information on the smoothness of results, to models that need only parameter values from data or existing literature. Thus, almost all models are grey box models as opposed to black box where no model form is assumed or white box models that are purely theoretical. (https://en.wikipedia.org/wiki/Grey_box_model)

In the following sections, the modeling structures progress from linear to polynomial to arbitrary nonlinear. In fact, in real world, nothing can be modeled as linear model, linear system only exists in human's mind as building blocks to construct complex nonlinear model. The moment we get into nonlinear real world, there is a learning element in modeling exercise.

While pursuing learning of nonlinearity, we as humans also want to maintain a big picture view so that we can focus on dominant effects not being boiled down by non-critical details. So at the end, I list several model simplification and model reduction methods.

Linear Regression

Simplest model structure is linear model, where we use linear regression to calculate the coefficients. Many Python packages offer ways to do linear regression, we can even do it ourselves by using the well-known formula

$$parameters = (X^T X)^{-1} X^T Y$$

I found the following Python package especially useful, in terms of selecting different variable set in X (http://statsmodels.sourceforge.net/).

```
import statsmodels.api as sm
from patsy import dmatrices
df=DataFrame(columns=['x1','x2','x3'])
y, X = dmatrices('y ~ x1+x2+x3', data=df, return_type='dataframe')
mod = sm.OLS(y, X)
res = mod.fit()
print res.summary()
```

Learning Nonlinearity: Spline Interpolation

In public domain, there are huge amount of design charts generated from design models. These charts can be readily reverse-engineered back to numerical models.

The first step is to use plot digitizer (e.g. webplotdigitizer http://arohatgi.info/WebPlotDigitizer/) to convert charts to data, then apply spline interpolation to rebuild the model numerically (https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html).

```
from scipy.interpolate import interp1d
fa = interp1d([6.0,6.5,7.0,7.5,8.0,8.5],[-4.3,-3.2,-2.2,-1.7,-0.7,-0.5])
fb = interp1d([6.0,6.5,7.0,7.5,8.0,8.5],[4.9,4.6,2.1,0.3,-0.4,-0.7])
print fa(6.2),fb(6.2)
-3.86 4.78
```

Learning Nonlinearity: Classification

Classification: K nearest neighbors (kNN) is one of the simplest learning strategies: given a new, unknown observation, look up in your reference database which ones have the closest features and assign the predominant class. (http://scikit-learn.org/stable/)

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[ 0.66666667  0.33333333]]
```

Classification and regression tree (CART) The models are obtained by recursively partitioning
the data space and fitting a simple prediction model within each partition.

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([ 0.5])
```

Learning Nonlinearity: Multivariate Adaptive Regression Splines (MARS)

MARS (https://en.wikipedia.org/wiki/Multivariate_adaptive_regression_splines) breaks nonlinearity into basis functions, while neural network (NN) in the next section breaks nonlinearity into neurons (this may remind you of Taylor series expansion or Fourier Series Expansion of any function)

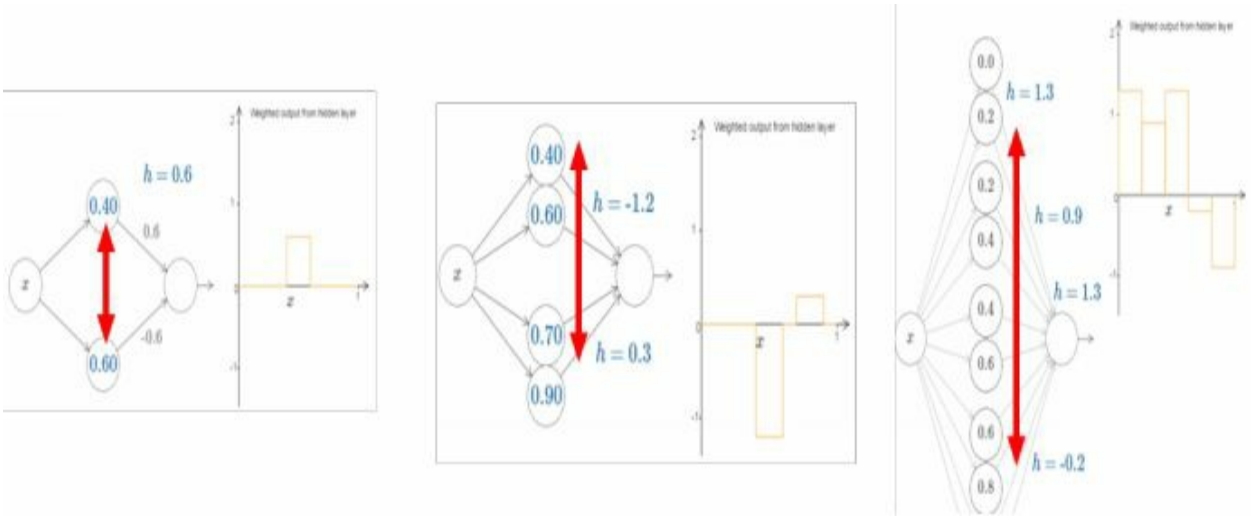
MARS model is a weighted sum of basis functions

```
import numpy
from pyearth import Earth
from matplotlib import pyplot

#Create some fake data
numpy.random.seed(0)
m = 1000
n = 10
X = 80*numpy.random.uniform(size=(m,n)) - 40
y = numpy.abs(X[:,6] - 4.0) + 1*numpy.random.normal(size=m)

#Fit an Earth model
model = Earth()
model.fit(X,y)

#Print the model
print(model.trace())
print(model.summary())
```

Learning Nonlinearity: Neural Network

Similarly, Neural Network (NN, https://en.wikipedia.org/wiki/Artificial_neural_network) model is a weighted sum of layers of neurons.

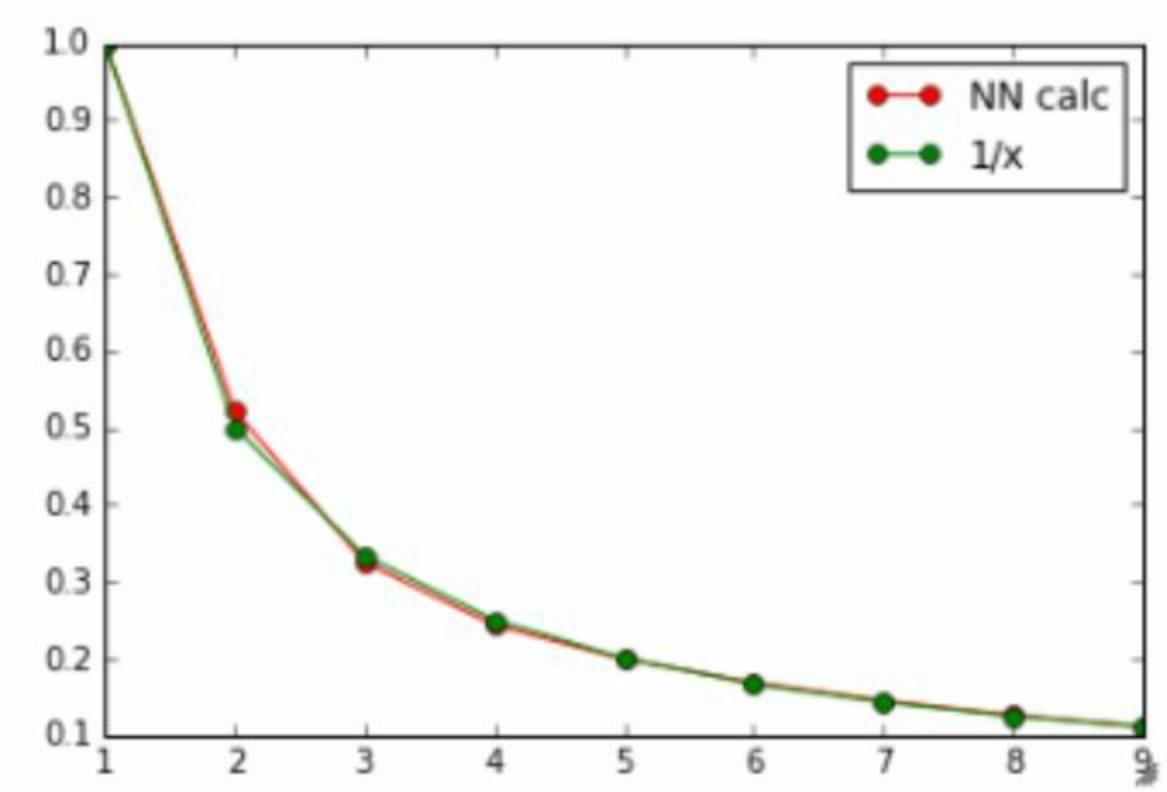
Universality tells us that neural networks can compute any function; and empirical evidence suggests that deep networks are the networks best adapted to learn the functions useful in solving many real-world problems. The following scheme shows a single layer of neurons can approximate any continuous nonlinearity by designing neurons to piecewise approximate the nonlinearity.



The following example shows a single layer NN can approximate y=1/x very well.

```
import numpy as np
def nonlin(x, deriv=False):
    if(deriv==True): return (x*(1-x))
    return 1/(1+np.exp(-x))

def NN_train(X,y):
    np.random.seed(1)
    syn0 = 2*np.random.random((np.shape(X)[1],np.shape(X)[0])) - 1
    syn1 = 2*np.random.random((np.shape(y)[0],np.shape(y)[1])) - 1
    #training step
    for j in xrange(60000):
        # Calculate forward through the network.
        l0 = X; l1 = nonlin(np.dot(l0, syn0)); l2 = nonlin(np.dot(l1, syn1))
```

```
        # Back propagation of errors using the chain rule.
        l2_error = y - l2; l2_delta = l2_error*nonlin(l2, deriv=True)
        if(j % 10000) == 0: print "Error: " + str(np.mean(np.abs(l2_error)))
        l1_error = l2_delta.dot(syn1.T); l1_delta = l1_error * nonlin(l1,deriv=True)

        #update weights (no learning rate term)
        syn1 += l1.T.dot(l2_delta); syn0 += l0.T.dot(l1_delta)
    return syn0,syn1

def NN_test(syn0,syn1,X):
    l0 = X; l1 = nonlin(np.dot(l0, syn0)); l2 = nonlin(np.dot(l1, syn1))
    return l2

def NN(fo,x_,x):
    #train
    bias=np.array([1.0]*len(x_))
    X_=np.array([x_,bias]).T
    y_=nonlin(fo(np.array([x_]).T))
    syn0,syn1=NN_train(X_,y_)

    #test
    bias=np.array([1.0]*len(x))
    X=np.array([x,bias]).T
    y=nonlin(fo(np.array([x]).T))
    y_test=NN_test(syn0,syn1,X)
    return y_test

%pylab inline
x_=np.array([1.0,2.5,5.0,7.5,10.0]);
x=np.arange(1.0,10.0,1);

fo=lambda x:1.0/(x)
y=fo(x)
y_test=NN(fo,x_,x)
Y=lambda y: -log(1.0/y-1)
plot(x,Y(y_test),'o-',c='r',label='NN calc'); legend(loc='best'); hold
plot(x,y,'o-',c='g',label='1/x'); legend(loc='best')
```


NN calc
1/x
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
1
2
3
4
5
6
7
8
9

Keep Big Picture: Nonlinear Transformation

Many natural phenomena, when gone through nonlinear transformation, can be formulated into linear equations.

Consider the following dynamic mass balance for CSTR chemical species C

$$V\frac{dC}{dt} = -OUT * C + IN * C_{IN} - k * V * C^{n}$$

At steady state

$$\frac{C}{C_{IN}} = N\left[1 - K * \left(\frac{C}{C_{IN}}\right)^{n}\right]$$

Where

$$N = \frac{IN}{OUT}, K = \frac{k * V * C_{IN}^{n-1}}{IN}$$

The parameterized regression equation is

$$y = N(1 - K * y^{n})$$

Or in its linear form

$$Y = a + b * X$$

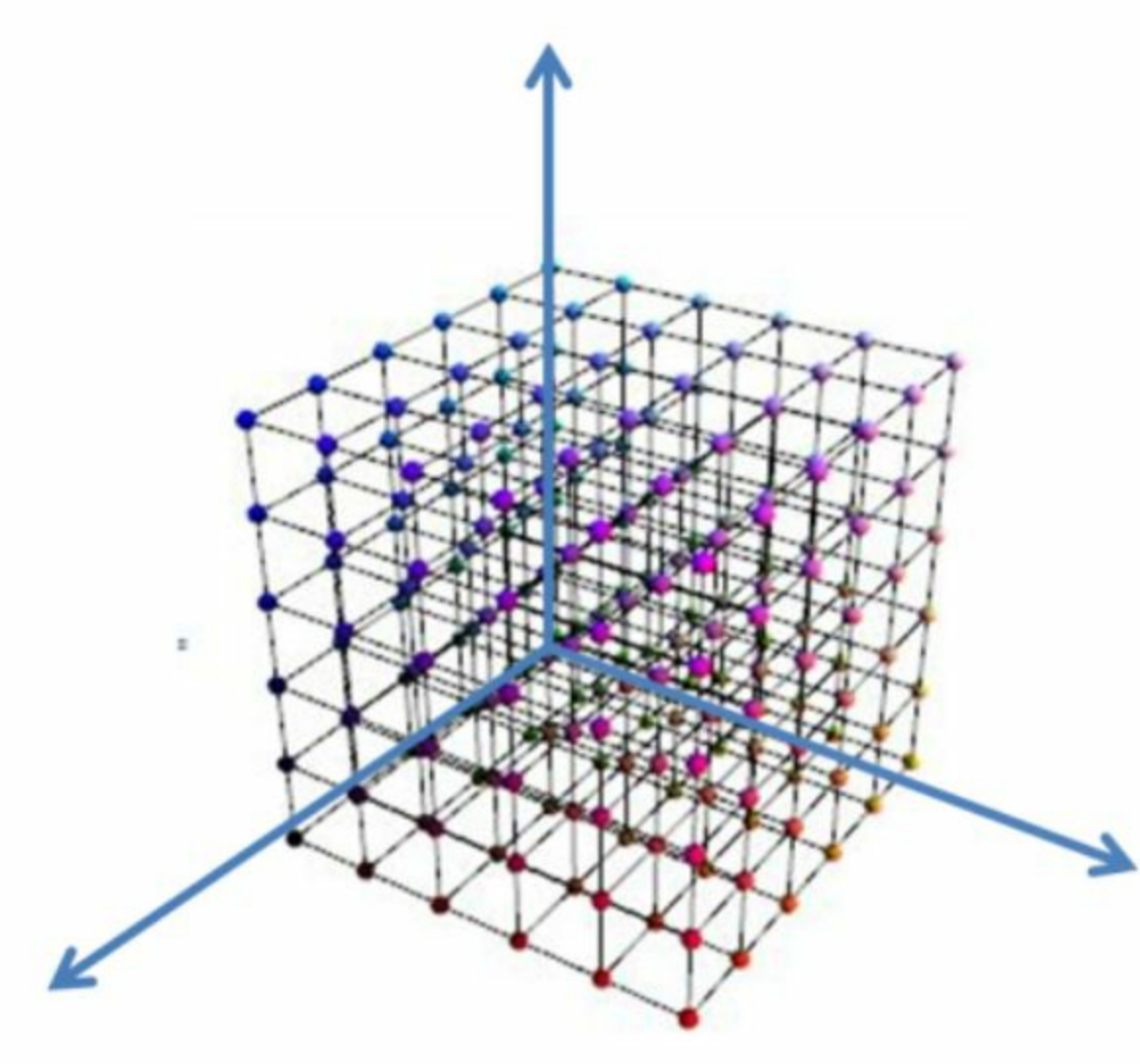
where

$$Y = \ln\left(\frac{1}{y} - \frac{1}{N}\right),\ y = \frac{C}{C_{IN}}, X = \ln(y), a = \ln(K), b = (n - 1)$$

As you can see, the above numbers are dimensionless. In fact, this is the most common approach an engineer would use, i.e. to define dimensionless qualities. Examples are Reynolds number as ratio of fluid inertial and viscous forces, Nusselt number as ratio of convective to conductive heat transfer,

Prandtl number as ratio of convective to conductive heat transfer, etc. (https://en.wikipedia.org/wiki/List_of_dimensionless_quantities) . These numbers characterize the ratios between competing natural forces, thus dramatically simplify nonlinearity into empirical equations of these dimensionless ratios, for which regression analysis can be carried out very easily.

Keep Big Picture: Surrogate Model



Even we have the physics design model, we can generate data grid over the entire state space by randomizing the grid using Monte Carlo simulation. With the data grid, we can use spline interpolation to rebuild a simplified surrogate

model, which can greatly facilitate visualization and animation.

Keep Big Picture: Model Reduction

Principle Component Analysis (PCA) is a dimension reduction technique that can find the combinations of variables that explain the most variance.

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
>>> print(pca.explained_variance_ratio_)
[ 0.99244...  0.00755...]
```

Clustering: Clustering groups together observations that are homogeneous with respect to a given criterion, finding "clusters" in the data.

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [4, 2], [4, 4], [4, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([0, 0, 0, 1, 1, 1], dtype=int32)
>>> kmeans.predict([[0, 0], [4, 4]])
array([0, 1], dtype=int32)
>>> kmeans.cluster_centers_
array([[ 1.,  2.],
       [ 4.,  2.]])
```

Balance between Forward Induction and Backward Induction

Application 1: Recommendation Engine

In a Recommendation Engine, the forward induction part is to study the similarity among the set, and the backward induction part is the recommendation based on similarity (https://www.youtube.com/watch?v=IjRy7dAWFdk). It is a perfect example for the law of effects, whose essential idea is that behavior can be modified by its consequences, i.e.
"responses that produce a satisfying effect in a particular situation become more likely to occur again in that situation, and responses that produce a discomforting effect become less likely to occur again in that situation."

Notation

- C is the set of customers in our domain. Its size is |C|.
- S is the set of services in our domain. Its size is |S|.
- S(c) is the set of services that customer c has rated.
- −S(c) is the complement of S(c) i.e., the set of services not yet seen by customer c.
- C(s) is the set of customers that have rated service s.
- −C(s) is the complement of C(s).

Goal of a recommendation system

$$\forall c \in C, s^* = argmax_{s \in S(c)}[R(c,s)]$$

Problem statement

The recommendation problem in its most basic form is that given a partially filled matrix of ratings or association scores (|C|x|S|), estimate the missing values

Methods

- Content-based filtering (a 'row-based' approach):

$$r_{c,s} = aggr_{s' \in S(c)}[r_{c,s'}]$$

- Collaborative filtering (a 'column-based' approach)

$$r_{c,s} = aggr_{c' \in C(s)}[r_{c',s}]$$

customer_id, service_id	c_1	c_2	c_3	c_4	c_5
s_1	?	?	4	?	1
s_2	3	?	?	2	2
s_3	3	?	?	?	?
s_4	?	1	2	1	1
s_5	?	?	?	?	?
s_6	2	?	2	?	?
s_7	?	?	?	?	?
s_8	3	1	5	?	?
s_9	?	?	?	?	2

Content-based filtering using mean ratings

```
def estimate1(customer_id, service_id):
    """ Simple content-filtering based on mean ratings. """
    return rating.ix[rating.customer_id == customer_id, 'rating'].mean()
print 'RMSE for estimate1: %s' % evaluate(estimate1)
```

Collaborative-based filtering using mean ratings

```
def estimate2(customer_id, service_id):
    """ Simple collaborative filter based on mean ratings. """
    ratings_by_others = rating[rating.service_id == service_id]
    if ratings_by_others.empty: return 3.0
    return ratings_by_others.rating.mean()
print 'RMSE for estimate2: %s' % evaluate(estimate2)
```

Generalized content-based filtering and collaborative-based filter can be achieved by weighting ratings with similarities among customers and services, respectively, i.e.

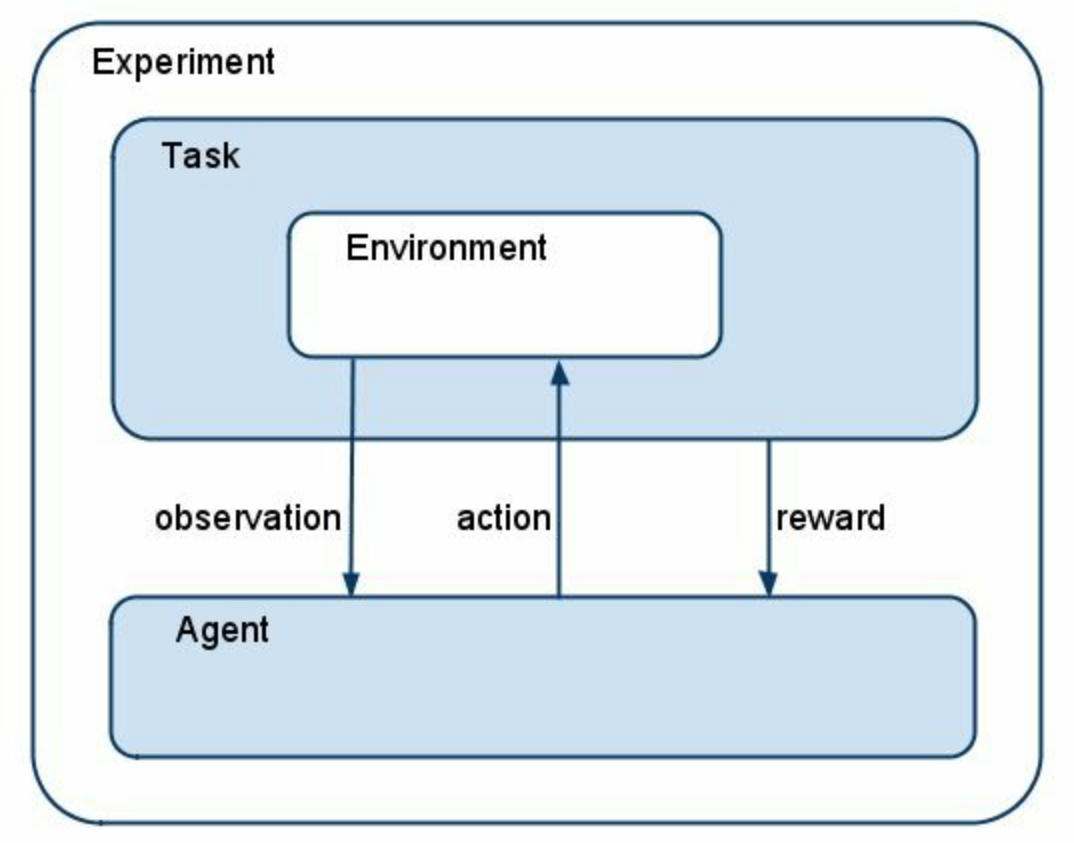
$$r_{c,s} = \frac{1}{\sum_{s' \in S(c)} |sim(s,s')|} \cdot \sum_{s' \in S(c)} sim(s,s') \cdot r_{c,s'}$$

for content-based or "row-based" filtering

$$r_{c,s} = \frac{1}{\sum_{c' \in C(s)} |sim(c,c')|} \cdot \sum_{c' \in C(s)} sim(c,c') \cdot r_{c',s}$$

for collaborative-based or "column-based" filtering

Application 2: Reinforcement Learning



In reinforcement learning (https://en.wikipedia.org/wiki/Reinforcement_learning), the forward induction and backward induction are "blended" into one very simple formula in the Q learning algorithm (https://en.wikipedia.org/wiki/Q-learning)

$$Q(state, action) = R(state, action) + Gamma * Max[Q(next\ state, all\ actions)]$$

Imaging there are N states, both Q and R are N*N matrices, they are used to represent transition from State I represented by row to State J by column. The entry in R is reward for each transition, set by the environment, the entry in Q is value for each transition, reflecting how the learning agent perceives reward R.

Each step, start from current State I in row, randomly select State J in column, the value update in Q matrix for this transition is the environment reward associated with transition from State I to State J (a forward induction or

exploration) plus weighted maximum value in Q for all possible next transitions from State J in row (a backward induction or exploitation).

The agent will explore from state to state until it reaches the goal. We'll call each exploration an episode. Each episode consists of the agent moving from the initial state to the goal state. Each time the agent arrives at the goal state, the program goes to the next episode.

The Gamma parameter has a range of 0 to 1 (0 <= Gamma > 1). If Gamma is closer to zero, the agent will tend to consider only immediate rewards. If Gamma is closer to one, the agent will consider future rewards with greater weight, willing to delay the reward.

The Q-Learning algorithm goes as follows:

1. Set the gamma parameter, and environment rewards in matrix R.
2. Initialize matrix Q to zero. For each episode: Select a random initial state. Do While the goal state hasn't been reached.
   - Select one among all possible actions for the current state.
   - Using this possible action, consider going to the next state.
   - Get maximum Q value for this next state based on all possible actions.
   - Compute: Q(state, action) = R(state, action) + Gamma * Max[Q(next state, all actions)]
   - Set the next state as the current state. End Do

   End For

```
def update_q(state, next_state, action, alpha, gamma):
    rsa = r[state, action]
    qsa = q[state, action]
    new_q = qsa + alpha * (rsa + gamma * max(q[next_state, :]) - qsa)
    q[state, action] = new_q
    # renormalize row to be between 0 and 1
    rn = q[state][q[state] > 0] / np.sum(q[state][q[state] > 0])
    q[state][q[state] > 0] = rn
    return r[state, action]
```

Application 3: Convex Optimization and Global Optimization

All analytics in the end is an optimization problem, and all optimization problem in the end is a search in a vector space or a function space, mathematically.

In math, abstraction progresses from a point in a number line, to a vector in a vector space, to a function in a function space.

In a function space, each dimension represents a basis function. In case of Tylor series expansion or Fourier series expansion, the function space has infinite number of dimensions for the basis functions.

So, optimization can be defined as searching a vector in a vector space such that gradient of a scalar function is zero, it can also be as searching a function in a function space such that gradient of the integral of the function is zero. The former uses "vector calculus", while the latter uses "calculus of variation". Moreover, for both, if independent variables are random variables (or variables w/ uncertainties), then we call them "stochastic calculus".

In optimization, one big question is whether the problem is convex or not. If it is convex, meaning only one global optimum, then any method that finds local optimum also finds the global optimum. Otherwise, there may exist multiple local optimums, methods that can only converge to local optimums, may not be able to find the global optimum.

Convex optimization is well developed, which is carried out in a backward induction approach, with optimal searching direction and search step size defined to reach the global optimum in high computational efficiency. Global optimization in case of multiple local optimums, is often carried in a forward induction approach, such as genetic algorithm, where a natural selection process that mimics biological evolution repeatedly modifies a population of individual solutions in search of the global optimum.

Convex optimization

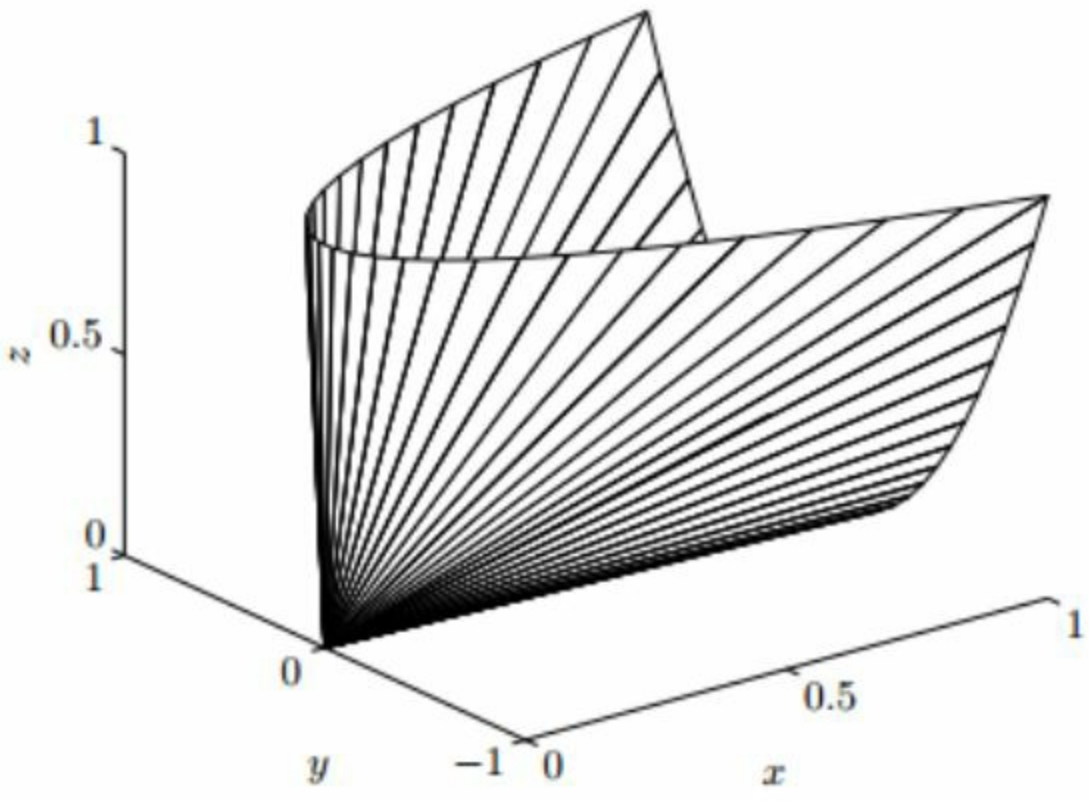
All convex optimization can be put as semidefinite programming (SDP, https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf), i.e.

$$\text{minimize } c^T x$$

$$\text{subject to } F(x) \geq 0$$

$$\text{where } F(x) = F_0 + \sum x_i F_i,\ F_0 \text{ and } F_i \text{ are symmetric}$$

For example, $F(x,y,z) = \begin{bmatrix} x & y \\ y & z \end{bmatrix} \geq 0$ is equivalent to $x \geq 0, z \geq 0, xz \geq y^2$, which can be visualized as a convex cone.



The positive definite (full-rank) matrices comprise the cone interior, while all singular positive semidefinite matrices (having at least one 0 eigenvalue) reside on the cone boundary. The only symmetric positive semidefinite matrix having all 0 eigenvalues resides at the origin.

CVXOPT (http://cvxopt.org/) is a free software package for convex optimization based on the Python programming language. It can be used with the interactive Python interpreter, on the command line by executing Python scripts, or integrated in other software via Python extension modules. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Python's extensive standard library and on the strengths of Python as a high-level programming language.

$$\text{minimize } x_1 - x_2$$

subject to

$$x_1\begin{bmatrix} -7 & -11 \\ -11 & 3 \end{bmatrix} + x_2\begin{bmatrix} 7 & -18 \\ -18 & 8 \end{bmatrix} \leq \begin{bmatrix} 33 & -9 \\ -9 & 26 \end{bmatrix}$$

```
from cvxopt import matrix, solvers
c = matrix([1.,-1.])
G = [ matrix([[-7., -11., -11., 3.], [ 7., -18., -18., 8.]]) ]
h = [ matrix([[33., -9.], [-9., 26.]]) ]
sol = solvers.sdp(c, Gs=G, hs=h)
>>> print(sol['x'])
```

CVXPY (http://www.cvxpy.org/en/latest/) is a Python-embedded modeling language for convex optimization problems. It allows you to express your problem in a natural way that follows the math, rather than in the restrictive standard form required by solvers. CVXPY relies on the open source solvers ECOS, CVXOPT, and SCS.

```
from cvxpy import *

# Create two scalar optimization variables.
x = Variable()
y = Variable()

# Create two constraints.
constraints = [x + y == 1,
               x - y >= 1]
```

```
# Form objective.
obj = Minimize(square(x - y))

# Form and solve problem.
prob = Problem(obj, constraints)
prob.solve()  # Returns the optimal value.
print "status:", prob.status
print "optimal value", prob.value
print "optimal var", x.value, y.value

status: optimal
optimal value 0.999999999762
optimal var 1.00000000001 -1.20824129796e-11
```

Optimization duality

Consider the following prime and dual optimization problems

$$\text{maximize } c^T x$$

$$\text{subject to } Ax \leq b,\ x \geq 0$$

And

$$\text{minimize } b^T y$$

$$\text{subject to } A^T y \geq c,\ y \geq 0$$

(https://en.wikipedia.org/wiki/Duality_(optimization)).

Think the first one as maximizing production as if you are the owner of the plant, subject to resource constraints, with x the production, c the profit per unit production, b the resource limits.

Think the second one as minimizing cost of manufacture facility as if you are buying the plant, subject to the constraints that selling a bundle of capacity that could be used to produce a product is at least as profitable as producing the product (http://web.stanford.edu/~ashishg/msande111/notes/chapter4.pdf), with y the shadow price (i.e. profit increase per unit increase of resource), $A^T y$ is product price. Intuitively, the total values for prime and dual problems are as follows

$$\sum_{j=1}^{n} c_j x_j + \sum_{i=1}^{m} y_i \left(b_i - \sum_{j=1}^{n} a_{ij} x_j \right),$$

and

$$\sum_{j=1}^{n}\left(c_j-\sum_{i=1}^{m}a_{ij}y_i\right)x_j+\sum_{i=1}^{m}b_iy_i,$$

respectively. In the end, the maximum profit pursued by the owner of the plant is identical to the minimum cost pursued by the buyer of the plant, both converge to the same evaluation of the plant.

Global optimization

For most nonlinear systems, where there may be multiple local optimums, we have to resort to global optimization, genetic algorithm is one of them, which is implemented in "Blackbox Optimization" in PyBrain package (http://pybrain.org/docs/index.html).

*"In a genetic algorithm, a population of candidate solutions to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties which can be mutated and altered. The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's properties is modified to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population". (*https://en.wikipedia.org/wiki/Genetic_algorithm *)*

In essence, various reinforcement learning algorithms are "genetic" algorithms, it is a natural selection on machine learning, thus a forward induction strategy.

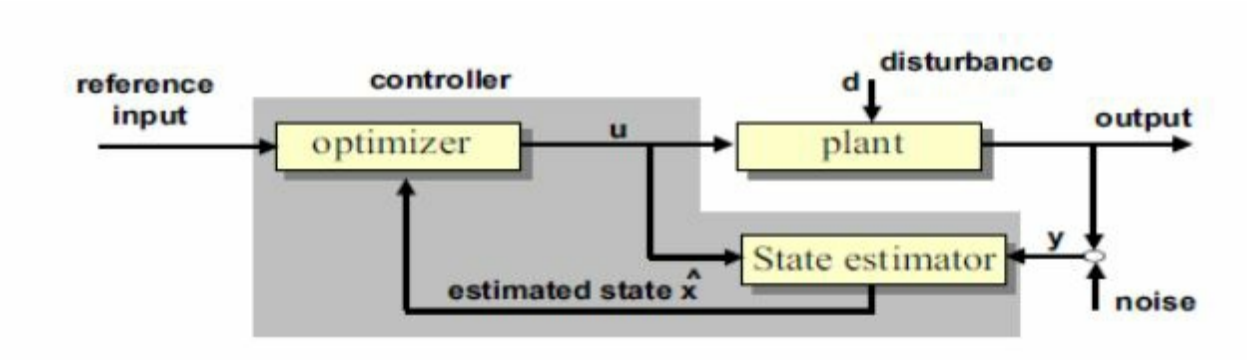
```
from pybrain.optimization import *
from scipy import randn
from pybrain.rl.environments.functions import SphereFunction
thetask = SphereFunction(3)
theparams = randn(3)
maxEvals = 1000
print 'HillClimber', HillClimber(thetask, theparams, maxEvaluations=maxEvals).learn()

HillClimber (array([-0.00332134,  0.01280978,  0.00315264]), 0.00018506100198510152)
```

Application 4: Model Identification, Optimal Control and State Estimation

Model control theory has 3 parts:

1. Model identification
2. Optimal state estimation
3. Optimal control

The first 2 parts are forward induction, while the 3rd part is backward induction. Both first 2 parts can blend into the execution of optimal control, making the control more robust and adaptive.



$$x(k+1) = Ax(k) + Bu(k)$$

$$y(k) = Cx(k)$$

Where

- A is State transition matrix,
- B is Control matrix,
- C is Observation matrix
- x is state
- u is input
- y is output

Model identification

Most commonly used model for time series analytics is Autoregressive–moving-average model with exogenous inputs model (ARMAX model, https://en.wikipedia.org/wiki/Autoregressive%E2%80%93moving-average_model). Given a time series of data Xt , the ARMAX model refers to

the model with autoregressive terms, moving average terms and exogenous inputs terms. This model contains

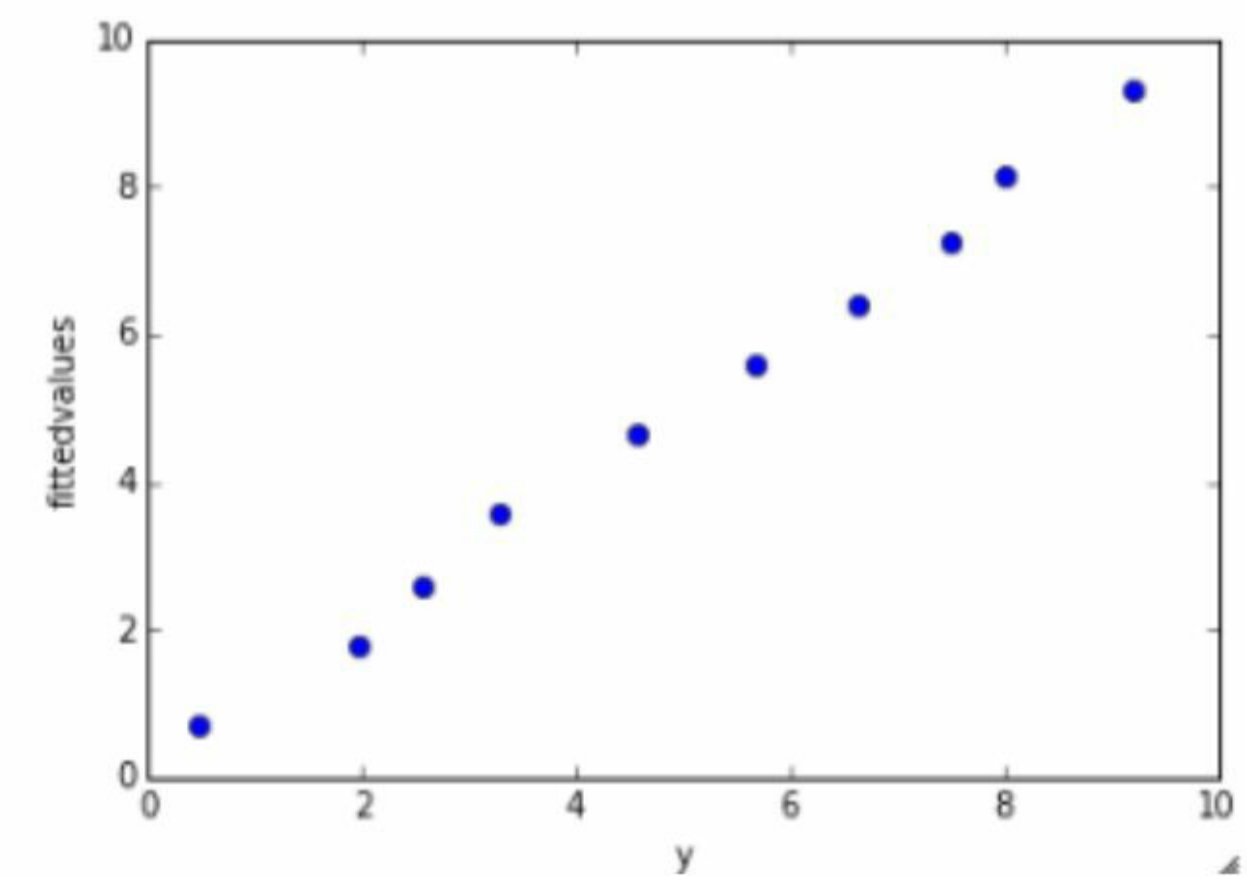
- an autoregressive (AR) part, which involves regressing the variable on its own lagged (i.e., past) values
- a moving average (MA) part, which involves modeling the error term as a linear combination of error terms occurring contemporaneously and at various times in the past, and
- a linear combination of the last terms of a known and external time series

For example,

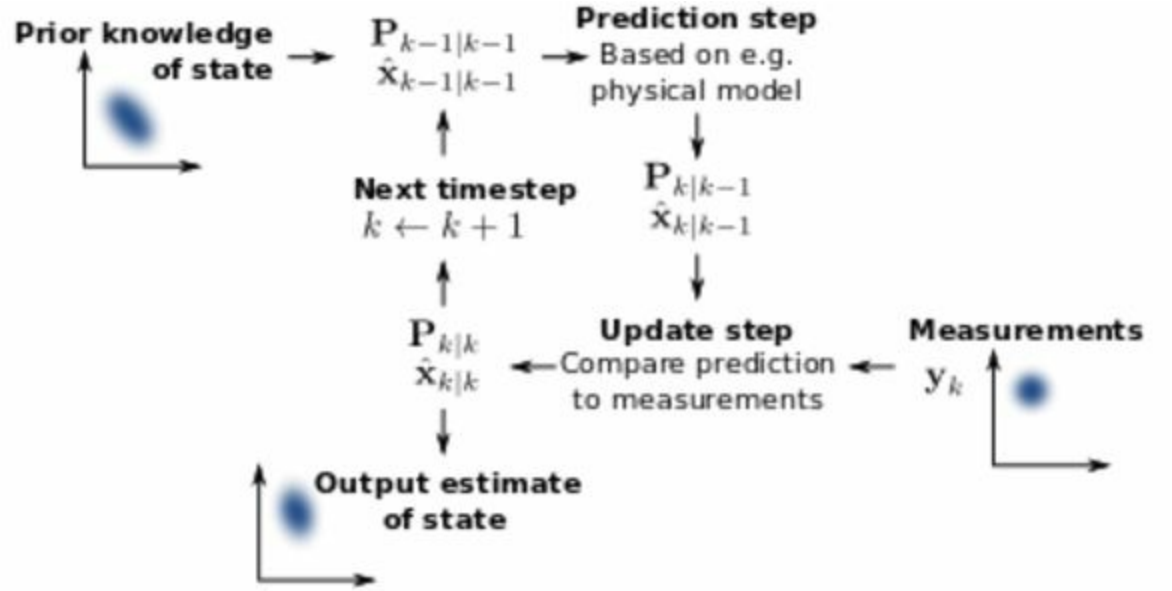
$$y[k] = - a1 * y[k-1] + b0 * u[k] + b1 * u[k-1] + c0 * e[k] + c1 * e[k-1]$$

solving the parameters is essentially a linear regression (http://statsmodels.sourceforge.net/stable/tsa.html).

```
import numpy as np
from scipy import stats
import pandas as pd
import statsmodels.api as sm

y = np.array(range(10))+np.random.rand(10)
u = np.array(range(10))
fit = sm.tsa.ARMA(y, (1,1),exog = u).fit()
plot(y,fit.fittedvalues,'o');
xlabel('y'); ylabel('fittedvalues')
```


10
8
6
4
2
0
fittedvalues
0
2
4
6
8
10
y

State estimation



Kalman filter (https://en.wikipedia.org/wiki/Kalman_filter) as well as other state estimation methods are in fact reinforcement learning, thus a forward induction method. For example, the Kalman filter keeps track of the estimated state of the system and the variance or uncertainty of the estimate. The estimate is updated using a state transition model and measurements. The value function is to minimize the variance or uncertainty of the estimation (http://greg.czerniak.info/guides/kalman1/)

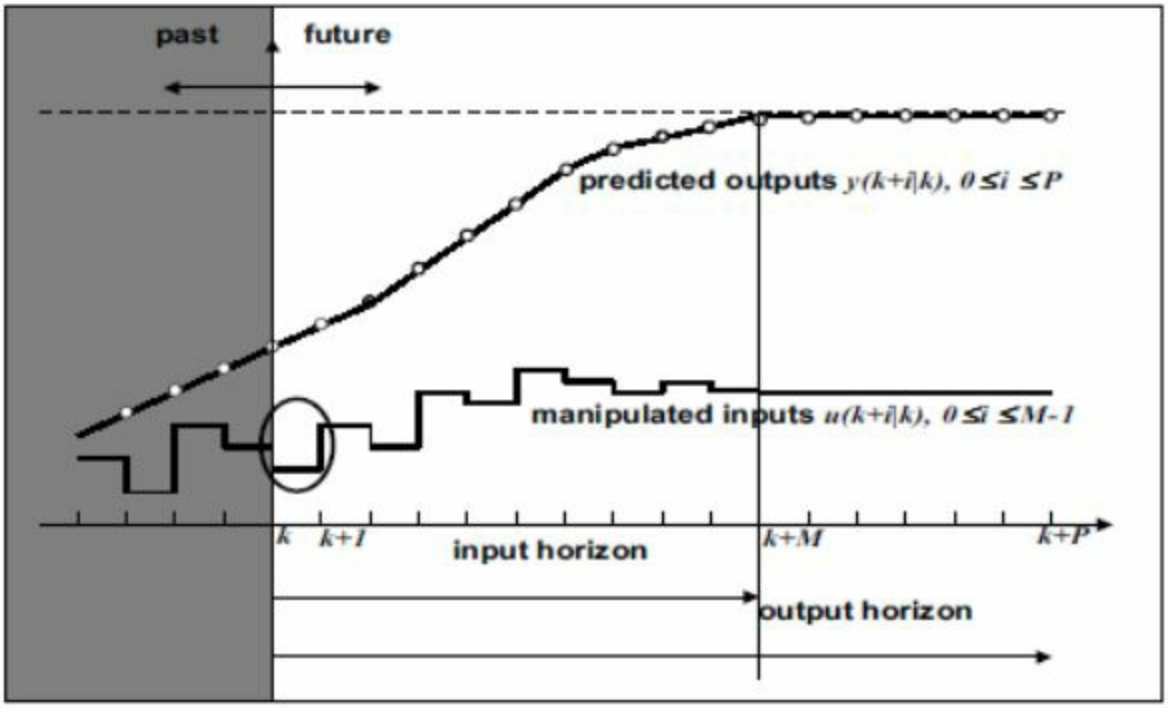
```
# Implements a linear Kalman filter.
class KalmanFilterLinear:
  def __init__(self,_A, _B, _C, _x, _P, _Q, _R):
    self.A = _A                  # State transition matrix.
    self.B = _B                  # Control matrix.
    self.C = _C                  # Observation matrix.
    self.current_state_estimate = _x# Initial state estimate.
    self.current_prob_estimate = _P  # Initial covariance estimate.
    self.Q = _Q                  # Estimated error in process.
    self.R = _R                  # Estimated error in measurements.
  def GetCurrentState(self):
    return self.current_state_estimate
  def Step(self,control_vector,measurement_vector):
    #---------------------------Prediction step-----------------------------
    predicted_state_estimate = self.A * self.current_state_estimate + self.B * control_vector
        predicted_prob_estimate = (self.A * self.current_prob_estimate) * numpy.transpose(self.A) \
                                          + self.Q
    #---------------------------Observation step-----------------------------
```

```
innovation = measurement_vector - self.C*predicted_state_estimate
innovation_covariance = self.C*predicted_prob_estimate*numpy.transpose(self.C) + self.R
#------------------------------Update step-------------------------------
    kalman_gain = predicted_prob_estimate * numpy.transpose(self.C) * \
                        numpy.linalg.inv(innovation_covariance)
self.current_state_estimate = predicted_state_estimate + kalman_gain * innovation
# We need the size of the matrix so we can make an identity matrix.
size = self.current_prob_estimate.shape[0]
      # eye(n) = nxn identity matrix.
      self.current_prob_estimate = (numpy.eye(size)-kalman_gain*self.C)*predicted_prob_estimate
```

Optimal control

Optimal control theory (https://en.wikipedia.org/wiki/Optimal_control) is based on calculus of variation. It searches in a function space a predicted output function and a manipulated input function such that a value integral of these two functions is maximized (or a cost integral of the two functions is minimized).

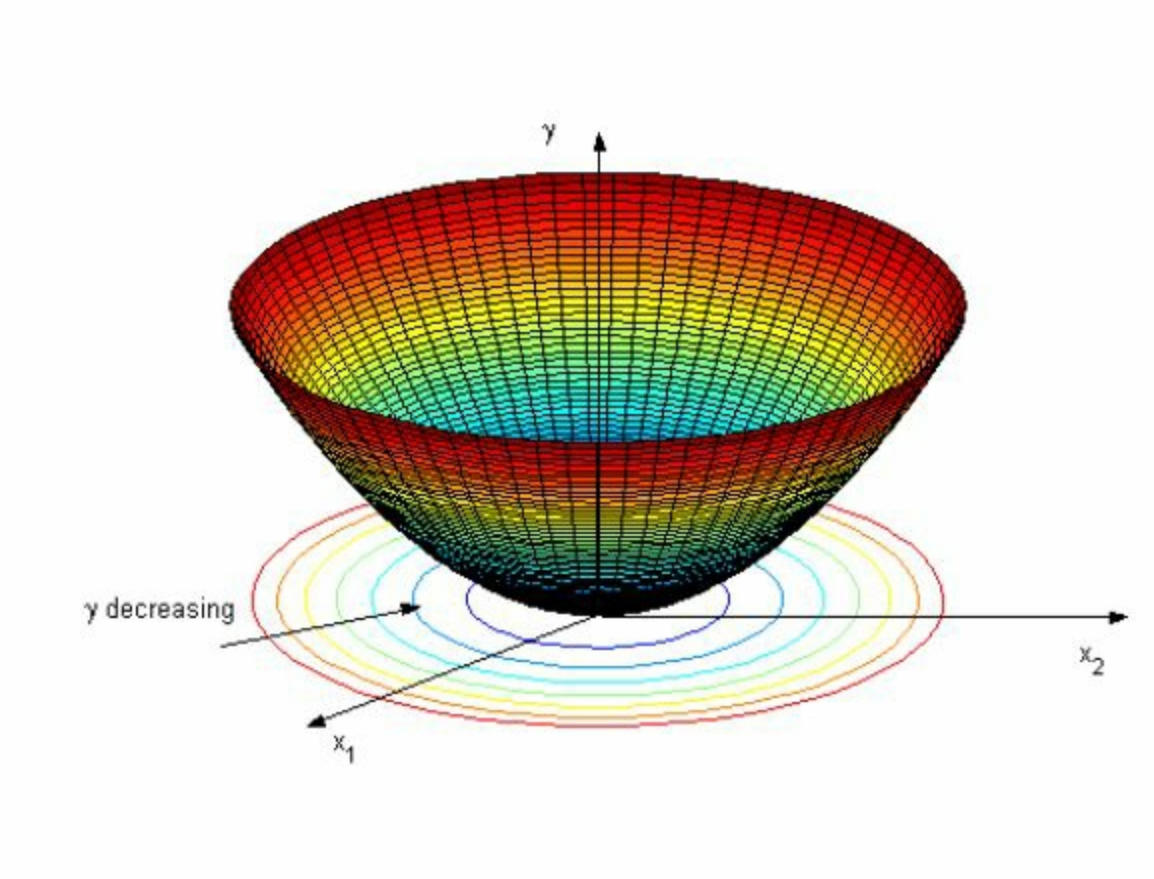
Model Predictive Control (MPC, https://en.wikipedia.org/wiki/Model_predictive_control) methodology is such a rea-time optimal control algorithm. At each sampling time, a dynamic optimization problem is solved. Process inputs are computed so as to optimize future plant behavior over a time interval known as the prediction horizon. A linear and/or quadratic objective function is used. Plant dynamics are described by an explicit process model. Process input and output constraints are incorporated directly into the problem formulation so that future constraint violations are anticipated and prevented. The first input of the optimal input sequence is applied to the plant and the problem is solved again at the next time interval using updated process measurements.



For a linear system, solving MPC or optimal control problem is a convex optimization problem. The optimization solves

- a Lyapunov function, which ensures control system stability or convergence to setpoint, and
- a feedback function, which minimizes the cost integral of predicted output function and manipulated input function.

The optimization can be solved very efficiently by semidefinite programming (as in section “convex optimization”).



One way of defining the Lyapunov function, which is always monotonically decreasing over time, is to define it as an upper bound of the cost integral of the predicted output function and manipulated input function over infinite horizon, i.e.

$$x(k)^T P x(k) \geq \sum (x(k)^T Q x(k) + u(k)^T R u(k))$$

Assume the following control policy

$$u(k) = Fx(k)$$

Then

$$x(k)^T P x(k) \geq \sum x(k)^T (Q + F^T R F) x(k)$$

As time progresses from k to k+1, the cost integral is always decreasing.

$$x(k)^T P x(k) - x(k+1)^T P x(k+1) \geq x(k)^T (Q + F^T R F) x(k)$$

So the optimization is as a semidefinite programming subject to linear matrix inequality constraints:

$$\text{minmize } x(k)^T P x(k)$$

subject to

$$P - (A + BF)^T P (A + BF) \geq Q + F^T R F$$

Next Generation Analytics: Analytics with "Free Will"

Having seen all the development in consumer markets, e.g. robots with artificial intelligence and visual/voice recognition, and connectivity among all the smart mobile devices, I am contemplating what it would be like for next generation analytics, and I can't help but wondering all these technologies will enable an analytics field that would soon possess the power of acting at its own discretion. To achieve that, an analytics system must have

- **wireless communications (**https://en.wikipedia.org/wiki/Wireless **) and mobile applications** (https://kivy.org/#home) that overcome physical limitations
- **robots** that sense from and exert actions to the world
- **competitive and learning analytics** that does compare-and-contrast to identify differentiation among existing analytics products and develop more competitive analytics products from data.

Summary

Product or Service

Throughout the book, I call the analytics software a product instead of a service, unlike the current fashion of calling everything “As a Service” (e.g. “Infrastructure As A Service”, “Platform As A Service”, “Software As A Service”). This is because I think “Product” represents a tighter integration of software, hardware and engineering, with the feature of easy use such as appliances. To achieve this tighter integration, we must be first and foremost an enthusiastic user of our own analytics product. Indeed, don’t we all want the use of a product as easy as a click of a button?

Open or Close

As you can see, throughout the book, the software I use are open source python packages. It is this openness that help ordinary developers to achieve the integration of software, hardware and engineering. We know that Apple is driving the close yet tightly integrated ecosystem paradigm, while Microsoft once deemed to drive the open ecosystem paradigm, is now deemed close, comparing to Google and vast open source ecosystems, such as Python. The force of open source ecosystem has, to large extent, displaced corporate IT. The open source ecosystem operates more like university professors getting funding to conduct research and publish results, only here the result is the software.

Technology and Humanity

I would say, whether the ecosystem is open or close, they are different means to the same end, i.e. to enable humanity to flourish. Apple's success is not so much because of its tight integration of software and hardware as its positioning itself at the intersection of technology and humanity. Its tight integration and control of ecosystem is because it emphasizes so much of human experience, human creativity in music and art, media and journalism.

In this book, I see the progressive improvement of analytics products is following along the same vein. For the second product, I add human cognition process to observe, identify, diagnose and improve the system. For the third product, I add modeling module to capture human knowledge, and use the balance between forward induction and backward induction to capture human reasoning/decision-making process. Finally, the very fact of publishing this book is my conviction that my effort of summarizing my learning can have some impacts on the generations to come.

Impact Analytics Product Development for the Better

When I did my programming, I found it is easier, simpler and more efficient to make modifications right at the "source of variation", for example, create more compact data structure, or process data per requirements right after reading out from database, or if I am doing too many patches at the later stage of the program, I would rather start a new structure and foundation instead of mending over a clumsy old structure. This is as I pointed before "***Be abstract***. *The ability to go broader in integration is based on the ability to go deeper in abstraction. Otherwise, you will be buried by the seemingly complex system*." In fact, I hope with Python, we, as developer, are more motivated to be abstract, therefore, help our users adopt more abstract thinking. Indeed, analytics is not yet another way of "machine replacing human", it should give smart and creative people more capability of doing abstract thinking thus be more powerful in making a better world.

Are We Better off with Internet of Things?

I think “Internet of Things” is a little bit misleading. I get it that it means connecting devices and letting devices talk to each other. Yet it fails to mention the integration between service providers and customers. My experience tells me that even one company can connect all the devices managed by itself, it may not provide added value to customer without connecting to customer devices, instead only improving the productivity and efficiency within the company. It is like playing a game, the collaborative outcome can only be achieved by sharing the information. Internet is for people to share things, the nature of the Internet of Things is still the same, i.e. for people to share information to achieve a common goal. Internet of Things does not automatically achieve value added, it is people’s willingness to overcome barriers and share the information that achieves added value.

Instead, Call It Analytic Field

Instead of following terminology that computer and Information technology experts use to reveal more connectivity among things/objects, I would follow the traditional physics approach and call it an Analytic field. “Things” are connected for the purpose of establishing this analytic field such that it can act upon all the things within the field, much like gravitation field, electromagnetic field, and quantum field acting forces on the objects within the fields. The Internet of things are indeed the field of people applying analytics to act upon things, it is human exerting their analytic power to make “things” as smart as human.

Bring in Stochastics and Uncertainty, Exploration and Learning

We can do as much as we can to model the physical world around us and make "optimal" decisions so long as the model matches reality, yet we have to admit, there are things we don't know that we don't know. This implies that there are stochastic variations and uncertainties that require us to explore and learn. Like many industries, risks and uncertainties are being formally quantified to trade with gains and returns, and learning becomes increasingly important to gain advantages in reducing the unknown risks and making smarter trades.

This is the Way to Ride Next Internet Revolution…

Do not rely the products that already lose their advantages and struggle to survive, instead, start from the basic knowledge of every aspect of integration and build a new world. The revolution comes from new generation, because each new generation must go through the basic cognitive process from infant to adulthood and learn the entire human knowledge again to be able to improve upon it. To ride this next wave of technology revolution, we need to abstract as broad and deep knowledge as possible, and rebuild the world. As Steve jobs said in his commencement talk at Stanford University “Stay Hungry. Stay Foolish.”