Engineering Analysis & Process Modeling

Dr. Prateek Kumar Jha

Assistant Professor, Department of Chemical Engineering

prateekjha.iitr@gmail.com

Recap: Equation Solving in one variable

All iterative methods starting with a guess value (two for Regula-Falsi method).
Iterate until $\left\|x^{(k+1)} - x^{(k)}\right\| < tolerance$

- Direct Substitution: Solve $x = f(x)$ using $x^{(k+1)} = f(x^{(k)})$
- Partial Substitution: Solve $x = f(x)$ using $x^{(k+1)} = x^{(k)} + \alpha\left(f\left(x^{(k)}\right) - x^{(k)}\right)$
  - $0 < \alpha < 1$ for under-relaxation → helps avoid oscillatory instability
  - $\alpha > 1$ for over-relaxation → may help in poor convergence

- Newton-Raphson method: Solve $F(x) = 0$ using

$$x^{(k+1)} = x^{(k)} - \frac{F\left(x^{(k)}\right)}{F'(x^{(k)})}$$

- Regula-Falsi method: Solve $F(x) = 0$ using

$$x^{(k+2)} = \frac{F\left(x^{(k)}\right)x^{(k+1)} - F\left(x^{(k+1)}\right)x^{(k)}}{F(x^{(k)}) - F(x^{(k+1)})}$$

Equation Solving in Many Variables: Vector form

$$\left.\begin{aligned} &F_1(x_1, x_2, \dots, x_n) = 0 \\ &F_2(x_1, x_2, \dots, x_n) = 0 \\ &\dots \\ &\dots \\ &F_n(x_1, x_2, \dots, x_n) = 0 \end{aligned}\right\} \quad \underline{\text{Vector form}}: \mathbf{F(X)} = \mathbf{0} \text{ where } \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ x_n \end{bmatrix} \text{ and } \mathbf{F(X)} = \begin{bmatrix} F_1(\mathbf{X}) \\ F_2(\mathbf{X}) \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ \cdot\cdot \\ F_n(\mathbf{X}) \end{bmatrix}$$

- If equations are linear, we can arrange it to a form

$$\mathbf{A} \cdot \mathbf{X} = \mathbf{B}$$

where $\mathbf{A}$ and $\mathbf{B}$ are matrices consisting of constants, and the solution is given by

$$\mathbf{X} = \mathbf{A^{-1}B}$$

Where $\mathbf{A^{-1}}$ is the matrix inverse of $\mathbf{A}$

- $\mathbf{A^{-1}}$ is not always easy to compute so we often use other direct methods (e.g., Gauss-elimination) or iterative methods (e.g., Gauss-Seidel)

Equation Solving in Many Variables: Jacobi Iteration

- Generalization of direct substitution for many variables
- Write $\mathbf{F}(\mathbf{X}) = \mathbf{0}$ in the form $\mathbf{X} = \mathbf{f}(\mathbf{X})$

$$x_1 = f_1(x_1, x_2, \dots, x_n)$$
$$x_2 = f_2(x_1, x_2, \dots, x_n)$$
$$\dots$$
$$\dots$$
$$x_n = f_n(x_1, x_2, \dots, x_n)$$

- Jacobi iteration: $\mathbf{X^{(k+1)}} = \mathbf{f}(\mathbf{X^{(k)}})$

Until $\|\mathbf{X^{(k+1)}} - \mathbf{X^{(k)}}\| < tolerance$. Here, $\|\cdot\|$ is the norm defined in many possible ways, e.g., the p-norm

$$\|\mathbf{X^{(k+1)}} - \mathbf{X^{(k)}}\| = \left(\sum_{i=1}^{n} \left| x_i^{(k+1)} - x_i^{(k)} \right|^p \right)^{\frac{1}{p}}$$

2-norm ($p = 2$) is commonly used.

Equation Solving in Many Variables: Newton-Raphson Method for Many Variables

$$\mathbf{X}^{(\mathbf{k+1})} = \mathbf{X}^{(\mathbf{k})} + \boldsymbol{\Delta}\mathbf{X}^{(\mathbf{k})}$$

$$\text{Until } \left\|\mathbf{X}^{(\mathbf{k+1})} - \mathbf{X}^{(\mathbf{k})}\right\| < tolerance.$$

$\boldsymbol{\Delta}\mathbf{X}^{(\mathbf{k})}$ is found by solving the linear equation

$$\mathbf{J}^{(\mathbf{k})} \cdot \boldsymbol{\Delta}\mathbf{X}^{(\mathbf{k})} = -\mathbf{F}(\mathbf{X}^{(\mathbf{k})})$$

Note that this equation is linear even when $\mathbf{F}(\mathbf{X})$ is nonlinear

$$\text{Jacobian, } \mathbf{J}^{(\mathbf{k})} = \begin{bmatrix} \frac{\partial F_1(\mathbf{X})}{\partial x_1} & \frac{\partial F_1(\mathbf{X})}{\partial x_2} & \cdots & \frac{\partial F_1(\mathbf{X})}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdots & \cdot\cdot \\ \frac{\partial F_n(\mathbf{X})}{\partial x_1} & \frac{\partial F_n(\mathbf{X})}{\partial x_2} & \cdots & \frac{\partial F_n(\mathbf{X})}{\partial x_n} \end{bmatrix} \text{ evaluated for } \mathbf{X} = \mathbf{X}^{(\mathbf{k})}$$

Equation Solving in

python™

for Chemical Engineers

Why Python?

- Python is an *Interpreted language*: saves the headache of compiling/linking (e.g., in C/C++)
- Easy to install, simple to use
  - Readable, intuitive command/function names
  - complex operations in a single statement
  - statement grouping is done by indentation instead of beginning and ending brackets
  - no variable or argument declarations are necessary
- But still a *real (high-level) programming language* (contain almost all of the features of advanced programming languages)
- Open-source (FREE!)
- Rich documentation: https://www.python.org/
- Library of functions for most of our mathematical needs apart from default very basic python functions, e.g., NumPy (basic functions), SciPy (advanced functions)
- Used by Google, Yahoo!,CERN, NASA

What is Ipython (Jupyter now!)?

IP[y]: Notebook Modulation Last Checkpoint: Jan 05 11:01 (autosaved)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

An angle modulated signal generally can be written as

$u(t) = A_c cos(2\pi f_c t + \phi(t))$

In a phase modulated (PM) system, the phase is proportional to the message

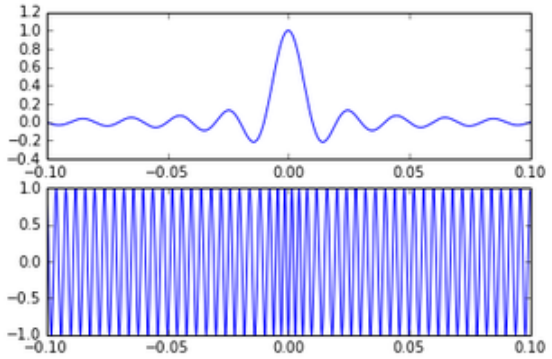
$\phi(t) = k_p m(t)$

In a frequency modulated (FM) system, instatantaneous frequency deviation is proportional to the message

$$f_i(t) - f_c = k_f m(t) = \frac{1}{2\pi}\frac{d}{dt}\phi(t)$$

In [12]:
```
from numpy.fft import fft,fftfreq
t = arange(-0.1,0.1,0.0001)
m = sinc(100*t)
int_m = empty(len(t))
for k in range(len(t)):
    int_m[k] = trapz(m[0:k],t[0:k])
u = cos(2*pi*250*t + 2*pi*100*int_m)
subplot(211)
plot(t,m)
subplot(212)
plot(t,u)
```

Out[12]: [<matplotlib.lines.Line2D at 0xd3a490c>]

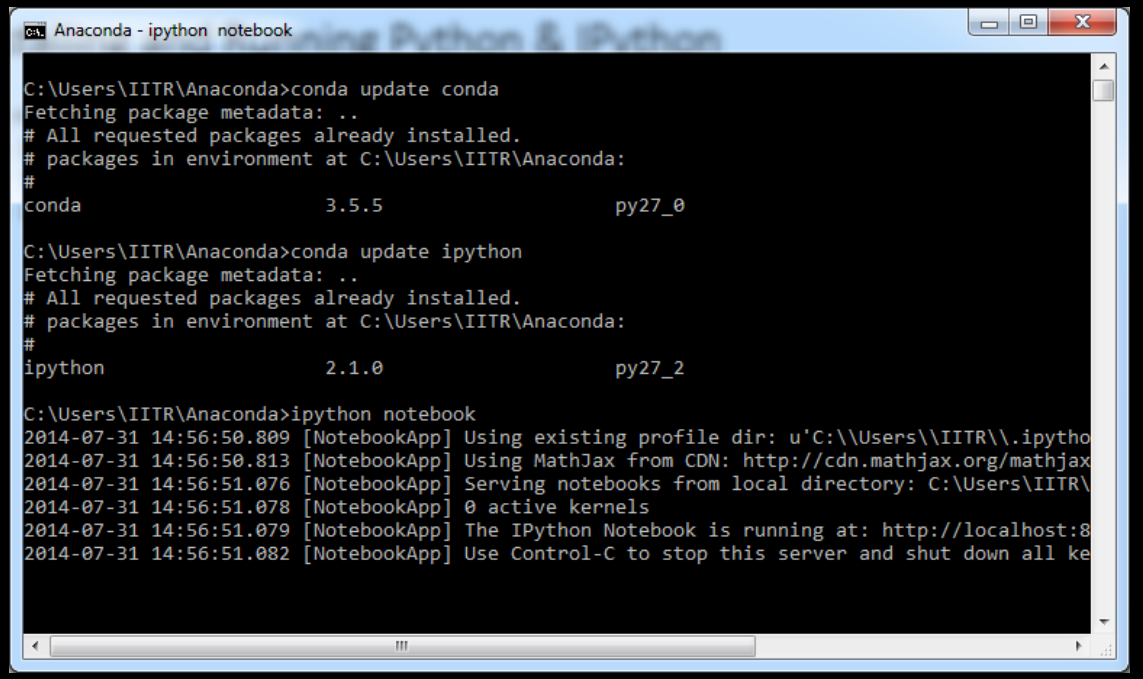


- A powerful interface for interactive computing using python
- Runs in a browser (e.g., Google Chrome)
- Notebooks (as MATLAB)

Installing and Running Python & Ipython (Jupyter)

For new users:

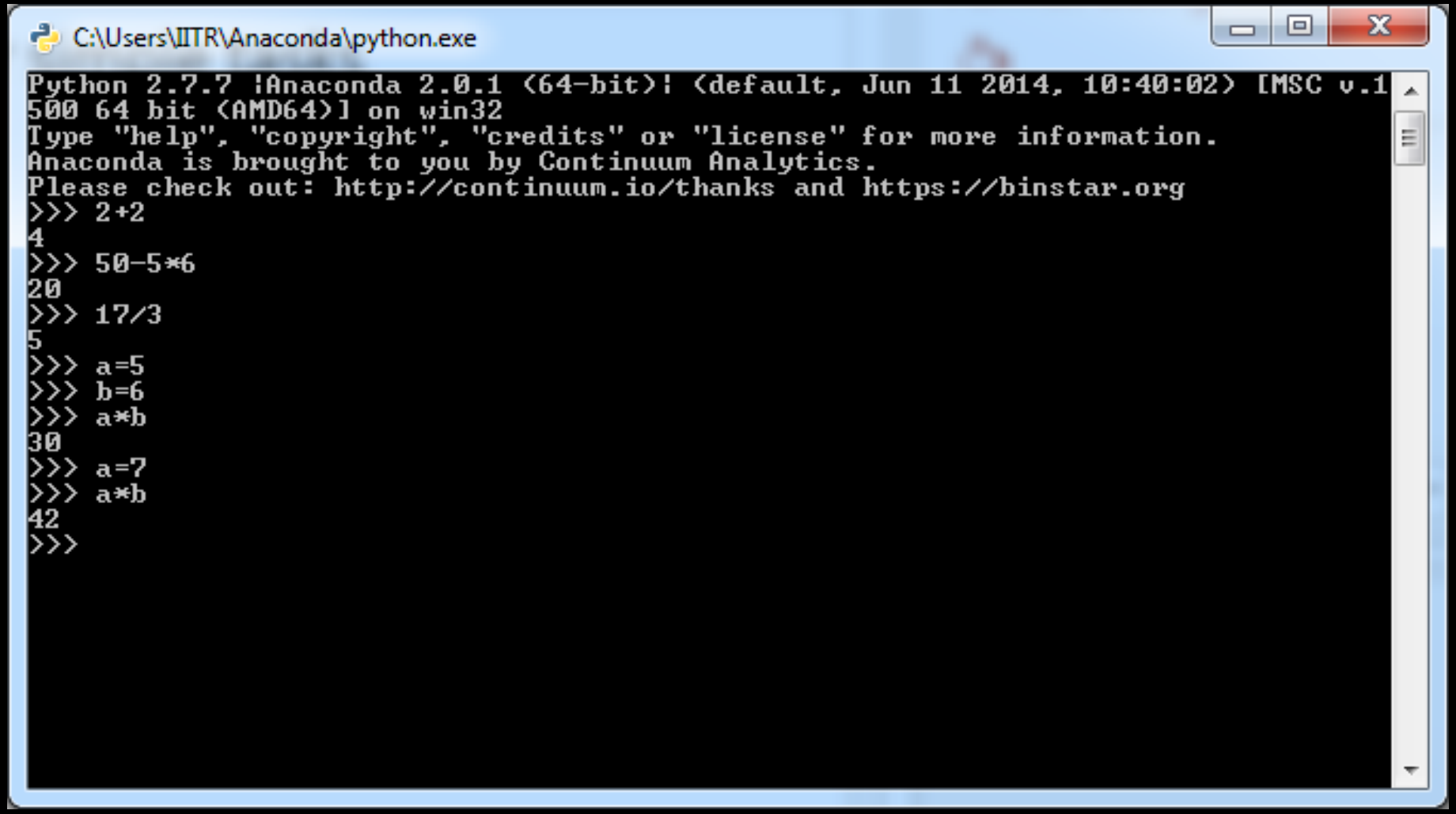
- Download and Install Anaconda (includes almost all libraries we need): http://continuum.io/downloads
- Update to the current version using the "Anaconda Prompt" and install jupyter
  - conda update conda
  - pip install jupyter



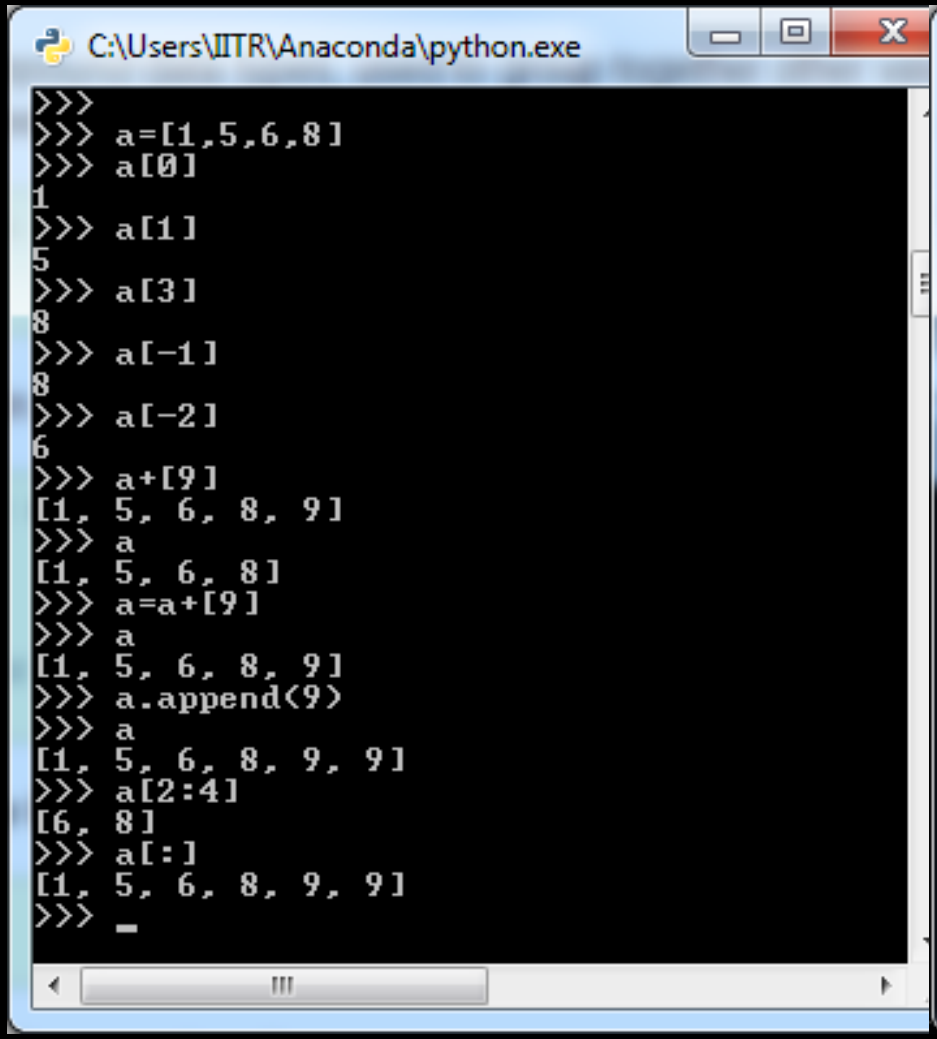
- Run ipython by typing "jupyter notebook" in the "Anaconda Command Prompt"
  Jupyter should open in your default browser (tested with Google Chrome)
- Create "New Notebook"

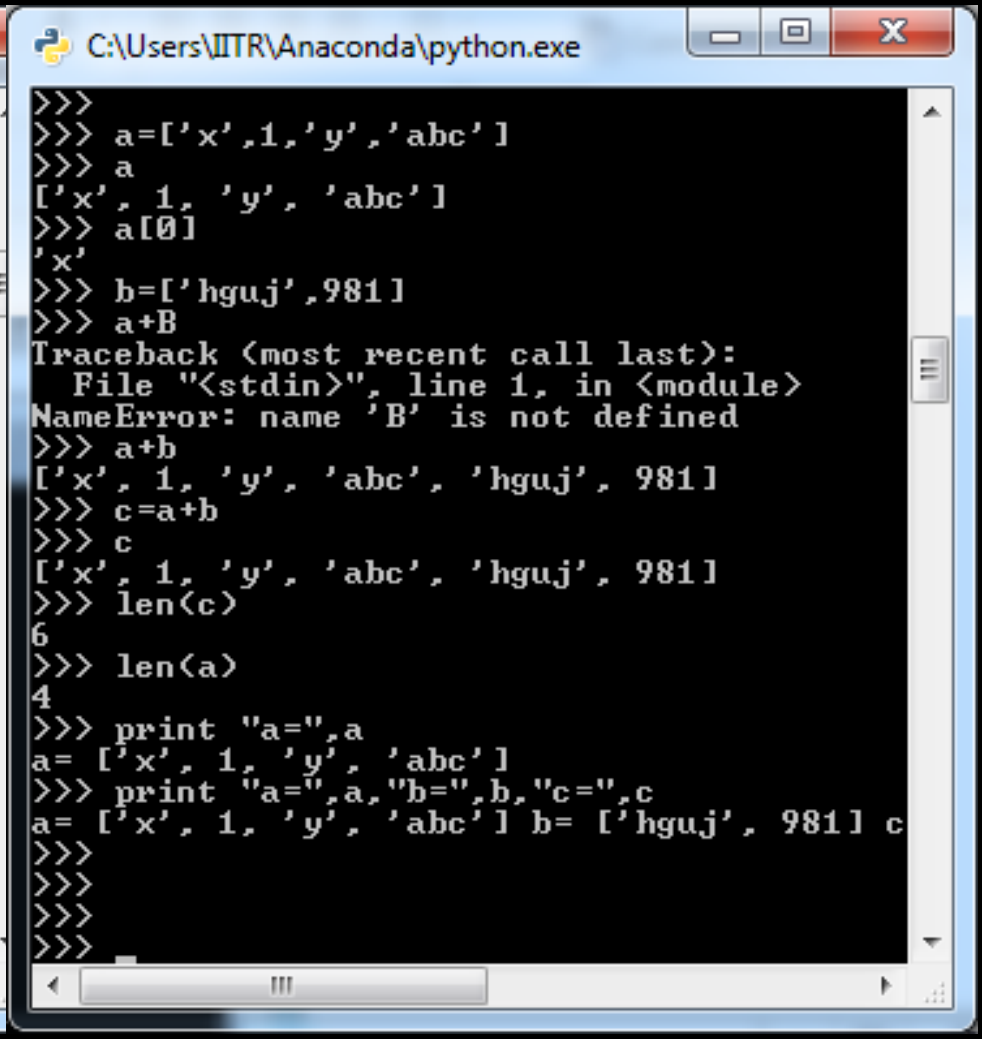
Shift+ENTER to run a block

You can also use python.exe for simple tasks (no need to create notebook)



Lists (simplest array type)





Direct substitution iteration can be written in the form

$$X_{new} = f(X)$$

- Consider the set of equations

$$3Y + 2X = 2 \qquad (1)$$

$$2Y + 3X = 4 \qquad (2)$$

- Algorithm for direct substitution

$$X_{new} = \frac{4 - 2Y}{3} = \frac{4}{3} - \frac{2}{3}\left(\frac{2 - 2X}{3}\right)$$

$$= \frac{8}{9} + \frac{4}{9}X = f(X)$$

Solution is reached when $X_{new} = X$ intersects $X_{new} = f(X)$

Jupyter code #1: Perform Direct Substitution and plots Xnew=f(X) and Xnew=X

Home | directsubstitution3

localhost:8888/notebooks/directsubstitution3.ipynb

jupyter directsubstitution3 (unsaved changes)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

In [1]:
```
#to make inline plots
%matplotlib inline
```

In [2]:
```
#Example of Direct Substitution
from numpy import fabs, linspace #import absolute value function, fabs from
import matplotlib.pyplot as plt #for plotting

Xold=3.0 #initial guess
error=1.0e9 #some large value
tolerance=1.0e-15

Xa=[] #empty arrays to store X values
fXa=[] #empty arrays to store f(X) values
```

12:16
31-01-2016

Home | directsubstitution3

localhost:8888/notebooks/directsubstitution3.ipynb

jupyter directsubstitution3 (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

Note the indent

```
while error>tolerance:
    print "X=", Xold
    Xnew=8.0/9.0+4.0/9.0*Xold #new value of X
    Xa.append(Xold) #append Xnew to Xa
    fXa.append(Xnew)    #append Y to Ya
    error=fabs(Xnew-Xold) #absolute value of error
    Xold=Xnew #set Xold to Xnew for next iteration

#Plotting
plt.xlabel('X')
plt.ylabel('Xnew')
plt.plot(Xa,fXa,'*-',Xa,Xa)
plt.legend(["Xnew=f(X)","Xnew=X"],loc="upper left")
plt.show()
```

12:18
31-01-2016

Home | directsubstitution3

localhost:8888/notebooks/directsubstitution3.ipynb

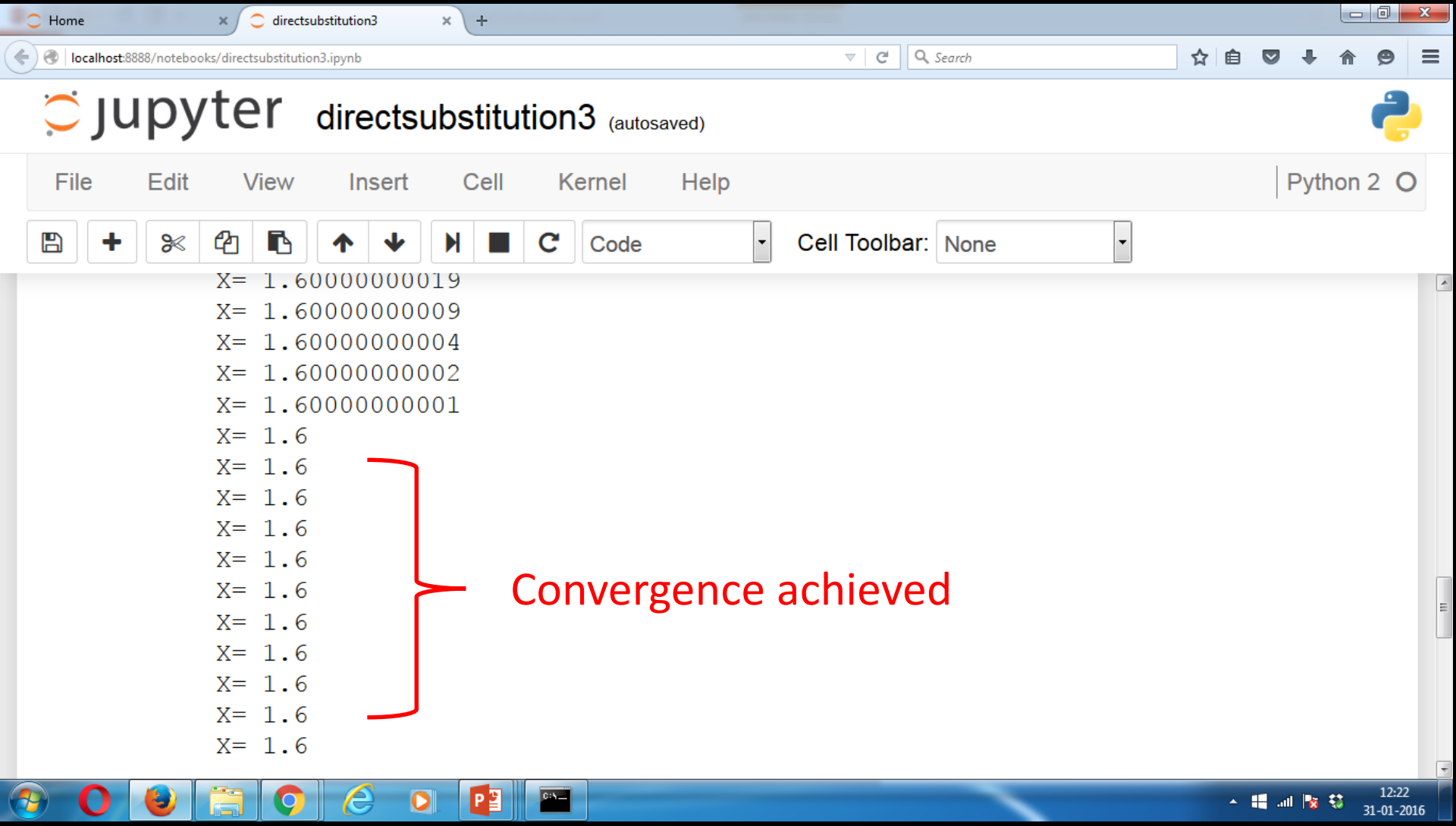
jupyter directsubstitution3 (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

```
X= 3.0
X= 2.22222222222
X= 1.87654320988
X= 1.72290809328
X= 1.65462581923
X= 1.62427814188
X= 1.61079028528
X= 1.60479568235
X= 1.60213141438
X= 1.60094729528
X= 1.60042102012
X= 1.60018712005
X= 1.60008316447
X= 1.60003696199
X= 1.60001642755
X= 1.60000730113
```

12:20
31-01-2016


Home
directsubstitution3
localhost:8888/notebooks/directsubstitution3.ipynb
Search
jupyter
directsubstitution3 (autosaved)
File
Edit
View
Insert
Cell
Kernel
Help
Python 2
Code
Cell Toolbar:
None
X= 1.60000000019
X= 1.60000000009
X= 1.60000000004
X= 1.60000000002
X= 1.60000000001
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
X= 1.6
Convergence achieved
12:22
31-01-2016

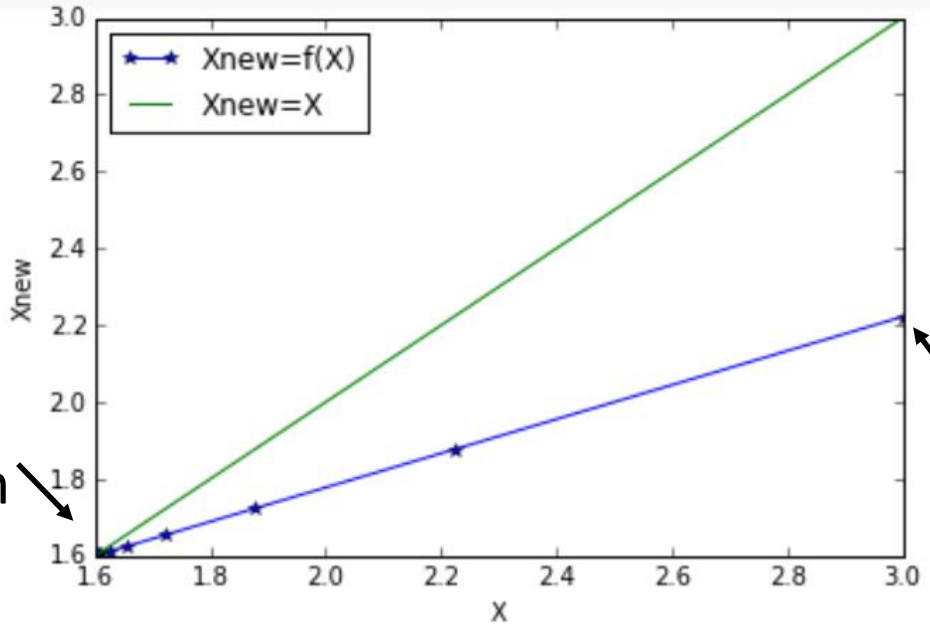
Home | directsubstitution3

localhost:8888/notebooks/directsubstitution3.ipynb

jupyter directsubstitution3 (autosaved)

File Edit View Insert Cell Kernel Help Python 2

Code Cell Toolbar: None

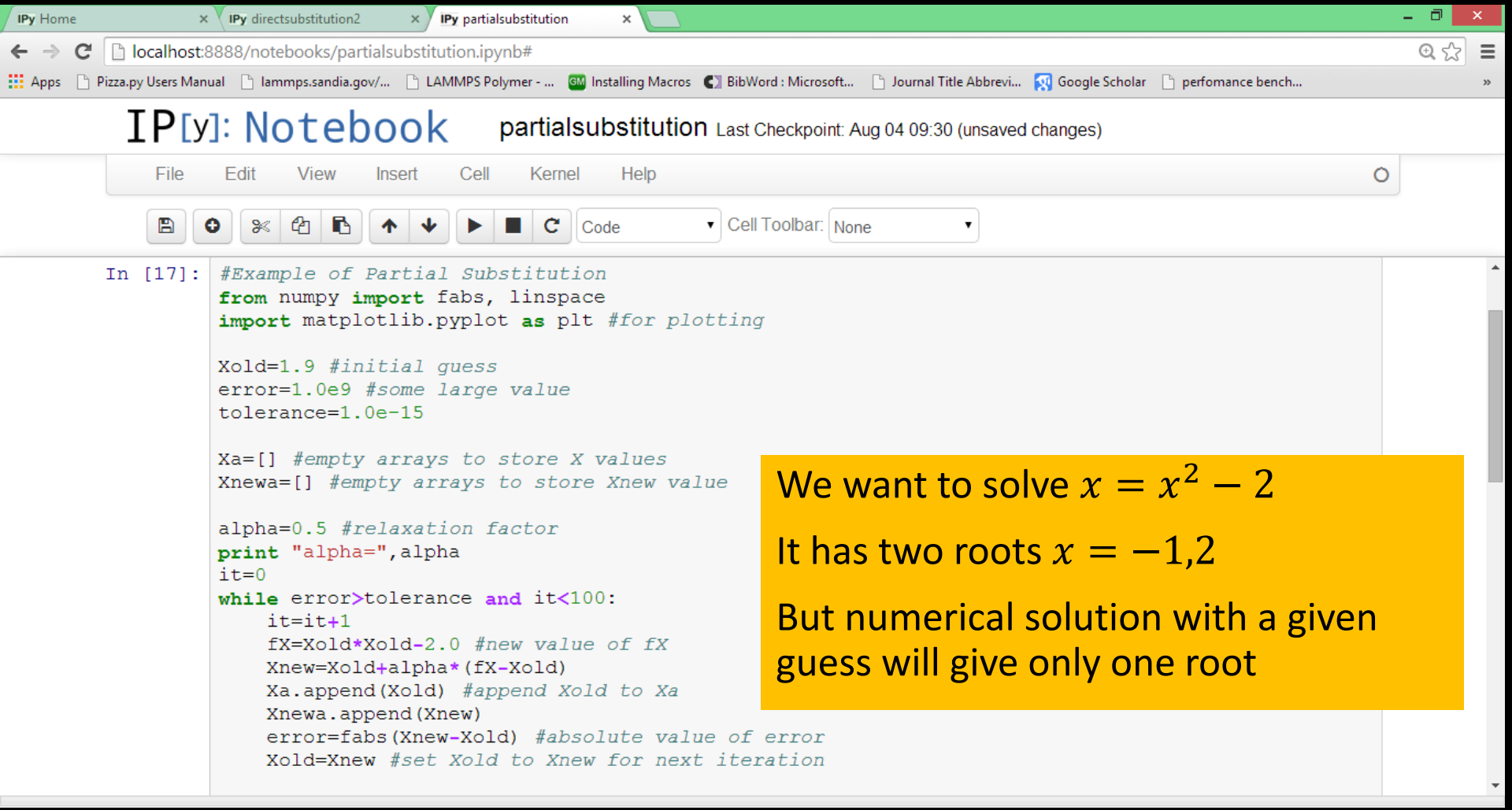


True Solution

Initial Guess

12:23
31-01-2016

Jupyter code #2: Partial substitution, $\alpha = 0.5$



We want to solve $x = x^2 - 2$

It has two roots $x = -1,2$

But numerical solution with a given guess will give only one root

IPy Home | IPy directsubstitution2 | IPy partialsubstitution

localhost:8888/notebooks/partialsubstitution.ipynb#

Apps | Pizza.py Users Manual | lammps.sandia.gov/... | LAMMPS Polymer - ... | Installing Macros | BibWord : Microsoft... | Journal Title Abbrevi... | Google Scholar | perfomance bench...

IP[y]: Notebook partialsubstitution Last Checkpoint: Aug 04 09:33 (unsaved changes)

File Edit View Insert Cell Kernel Help

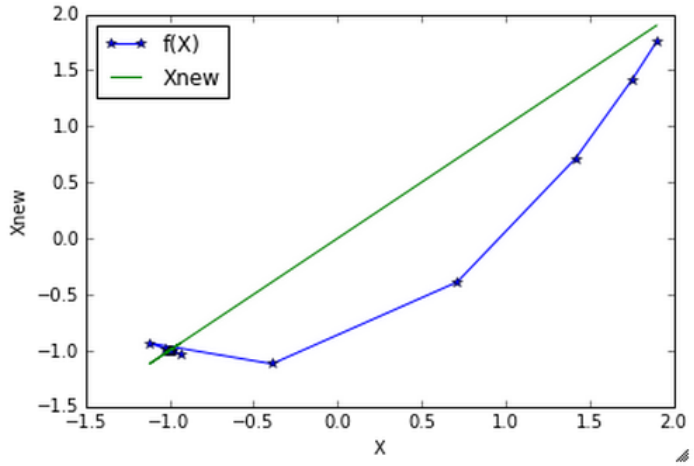
Code Cell Toolbar: None

```
print "alpha=",alpha
it=0
while error>tolerance and it<100:
    it=it+1
    fX=Xold*Xold-2.0 #new value of fX
    Xnew=Xold+alpha*(fX-Xold)
    Xa.append(Xold) #append Xold to Xa
    Xnewa.append(Xnew)
    error=fabs(Xnew-Xold) #absolute value of error
    Xold=Xnew #set Xold to Xnew for next iteration

if error<=tolerance:
    print "Convergence reached in ", it, " iterations"
else:
    print "Convergence not achieved in ", it, "iterations"

#Plotting
plt.xlabel('X')
plt.ylabel('Xnew')
plt.plot(Xa,Xnewa,'*-',Xa,Xa)
plt.legend(["f(X)","Xnew","Xnew=X"],loc="upper left")
plt.show()
```

alpha= 0.5

```
plt.plot(Xa,Xnewa,'*-',Xa,Xa)
plt.legend(["f(X)","Xnew","Xnew=X"],loc="upper left")
plt.show()
```

```
alpha= 0.5
Convergence reached in  54  iterations
```



Jupyter code #3: earlier code with alpha=1 (direct substitution)

IPy Home | IPy directsubstitution2 | IPy partialsubstitution

localhost:8888/notebooks/partialsubstitution.ipynb#

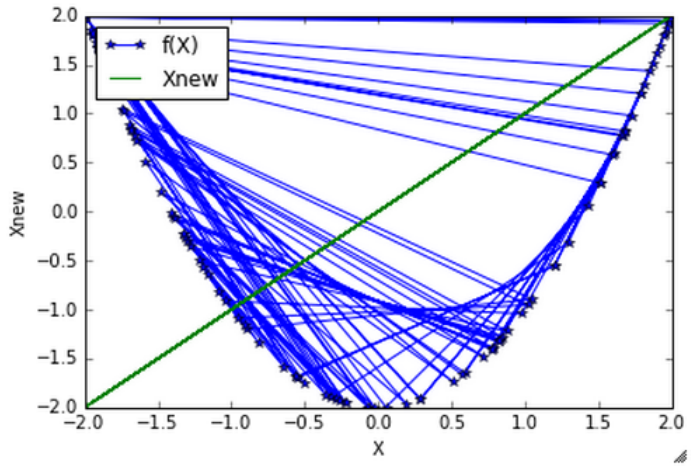
Apps | Pizza.py Users Manual | lammps.sandia.gov/... | LAMMPS Polymer - ... | Installing Macros | BibWord : Microsoft... | Journal Title Abbrevi... | Google Scholar | perfomance bench...

IP[y]: Notebook partialsubstitution Last Checkpoint: Aug 04 09:33 (unsaved changes)

File Edit View Insert Cell Kernel Help

Code Cell Toolbar: None

```
alpha= 1.0
Convergence not achieved in  100 iterations
```



In [19]: