Comparing Python, MATLAB, and Mathcad

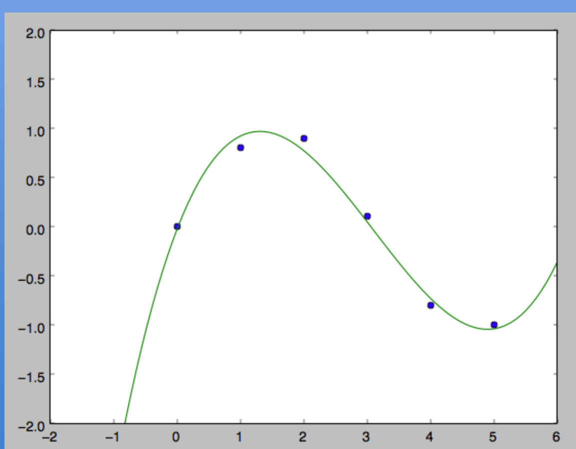
- Sample Code in Python, Matlab, and Mathcad
  - Polynomial fit
  - Integrate function
  - Stiff ODE system
  - System of 6 nonlinear equations
  - Interpolation
  - 2D heat equation: MATLAB/Python only
- IPython Notebooks

Thanks to David Lignell for providing the comparison code

Code: fit polynomial to data

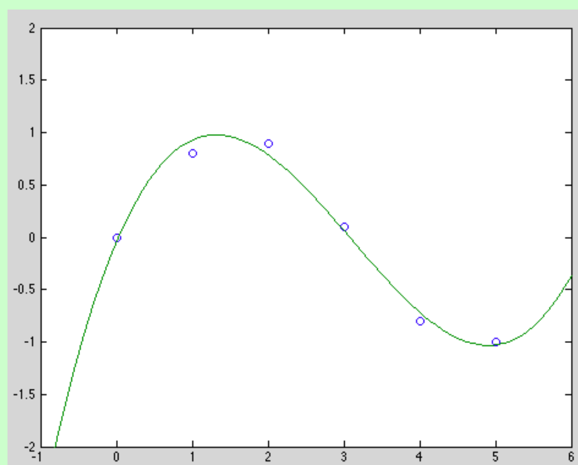
Python

```python
from numpy             import *
from scipy.interpolate import *
from matplotlib.pyplot import *

x  = array([0, 1, 2, 3, 4, 5])
y  = array([0, 0.8, 0.9, 0.1, -0.8, -1])
xp = linspace(-2,6,100)

p3 = polyfit(x,y,3)

plot(x,y,'o', xp,polyval(p3,xp),'-')
ylim(-2,2)
ion(); show()
```



Matlab

```matlab
x  = [0, 1, 2, 3, 4, 5];
y  = [0, 0.8, 0.9, 0.1, -0.8, -1];
xp = linspace(-2,6,100);

p3  = polyfit(x,y,3);

plot(x,y,'o', xp,polyval(p3,xp),'-');
ylim([-2,2]);
```



Mathcad

$x := (0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5)^T$

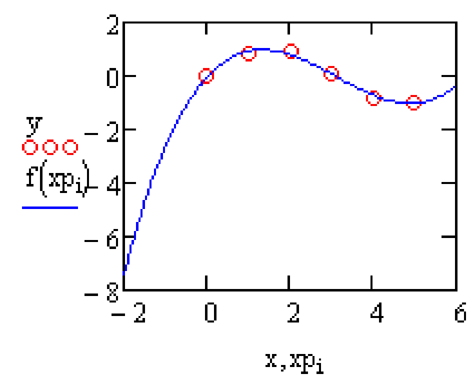
$y := (0\ \ 0.8\ \ 0.9\ \ 0.1\ \ -0.8\ \ -1)^T$

$i := 0..99$

$xp_i := i \cdot \frac{8}{99} - 2$

$p3 := \text{regress}(x,y,3)$

$f(xp) := \text{interp}(p3,x,y,xp)$



Code: integrate function

Python

```
from numpy                import *
from scipy.integrate      import *
from matplotlib.pyplot import *

def F(x) :
    return 2*x**2 + 1

I = quad(F,0,1)

print I
```

Matlab

```
I = quad(@integrationF, 0,1)
```

```
function F = integrationF(x)

    F = 2*x.^2 + 1;

end
```

2 files (optional)

Mathcad

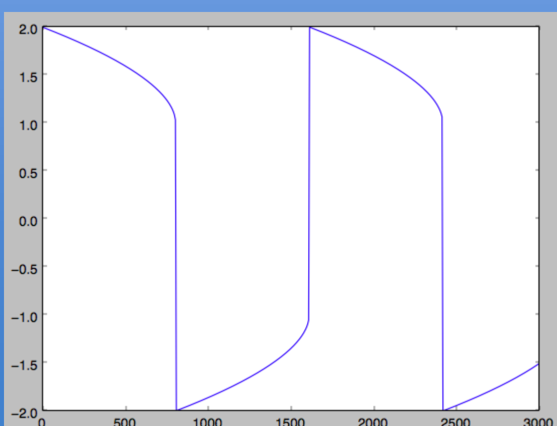
$$f(x) := 2 \cdot x^2 + 1$$

$$\int_0^1 f(x)\,dx = 1.667$$

Code: Stiff ODE system

Python

```python
from numpy              import *
from scipy.integrate    import *
from matplotlib.pyplot import *

def odeF(y,t) :
    dydt = zeros(2)
    dydt[0] = y[1]
    dydt[1] = 1000*(1-y[0]**2)*y[1]-y[0]
    return dydt

t = linspace(0,3000,500)
y0 = array([2, 0])
y = odeint(odeF, y0, t, mxstep=1000)

plot(t,y[:,0],'-')
ion(); show()
```

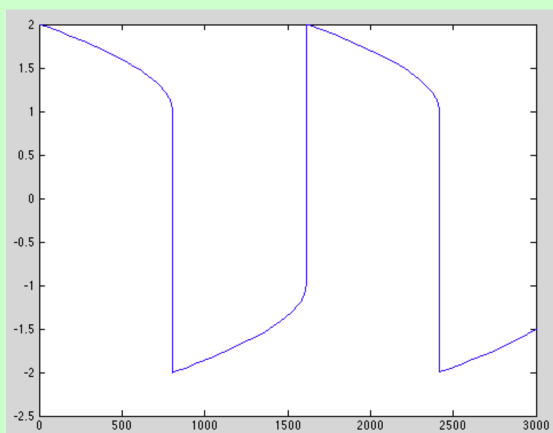


Matlab

```matlab
function ydot = odeF(t,y)

    ydot(1,1) = y(2);
    ydot(2,1) = 1000*(1-y(1)^2)*y(2)-y(1);

end
```

```matlab
tend = 3000;
y0   = [2 0];
[t y] = ode15s(@odeF, [0 tend], y0);

plot(t,y(:,1),'-');
```

2 files



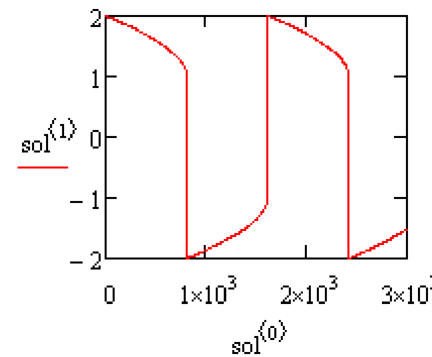
Mathcad

$$y0 := \begin{pmatrix} 2 \\ 0 \end{pmatrix}$$

$$D(t,y) := \begin{bmatrix} y_1 \\ 1000 \cdot \left[1 - (y_0)^2\right] \cdot y_1 - y_0 \end{bmatrix}$$

$$tend := 3000$$

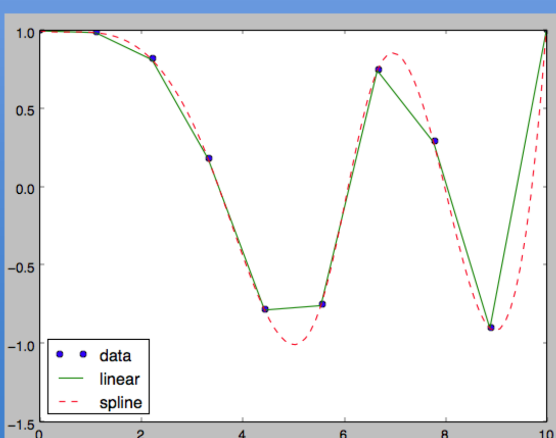
$$sol := AdamsBDF(y0,0,tend,500,D)$$



Code: interpolation

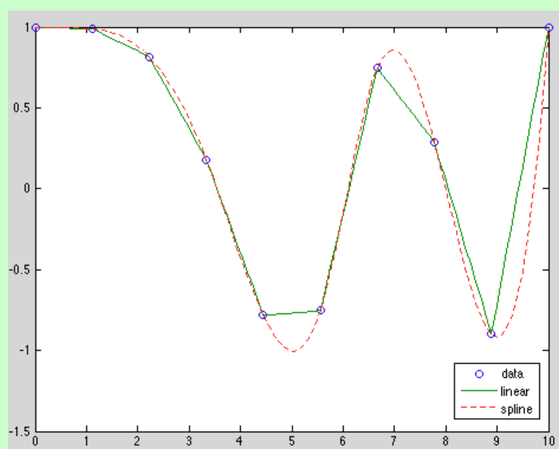
Python

```python
from numpy                import *
from scipy.interpolate import *
from matplotlib.pyplot import *

x1          = linspace(0,10,10)
y1          = cos(x1**2/8)
x2          = linspace(0,10,100)

f_linear  = interp1d(x1,y1)
f_spline  = interp1d(x1,y1, kind='cubic')
y2_linear = f_linear(x2)
y2_spline = f_spline(x2)

plot(x1,y1,'o', x2,y2_linear,'-', x2,y2_spline,'--')
legend(['data', 'linear', 'spline'], loc='best')
ion(); show()
```



Matlab

```matlab
x1        = linspace(0,10,10);
y1        = cos(-x1.^2/8);
x2        = linspace(0,10,100);

y2_linear = interp1(x1,y1,x2,'linear','extrap');
y2_spline = interp1(x1,y1,x2,'spline','extrap');

plot(x1,y1,'o', x2,y2_linear,'-', x2,y2_spline,'--')
legend('data', 'linear', 'spline', 'Location','Best')
```



Mathcad

$i := 0..9$

$x1_i := i \cdot \frac{10}{9}$

$y1_i := \cos\left[\frac{-\left(x1_i\right)^2}{8}\right]$

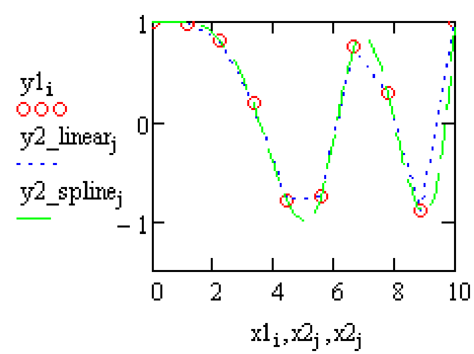
$j := 0..99$

$x2_j := \frac{j \cdot 10}{99}$

y2_linear := linterp(x1, y1, x2)

f_spline := cspline(x1, y1)

y2_spline := interp(f_spline, x1, y1, x2)



Code: system of nonlinear equations

Python

```python
from scipy.optimize import *
from numpy          import *

#--------------------------------------

def solverF(x) :

    f1 = x[0];
    f2 = x[1];
    f3 = x[2];
    Q1 = x[3];
    Q2 = x[4];
    Q3 = x[5];

    Qt = 0.01333;         # Given total volumetric flow rate
    e1 = 0.00024;         # pipe roughness (m)
    e2 = 0.00012;
    e3 = 0.0002;
    L1 = 100;             # pipe length (m)
    L2 = 150;
    L3 = 80;
    D1 = 0.05;            # pipe diameter (m)
    D2 = 0.045;
    D3 = 0.04;
    mu = 1.002E-3;        # viscosity (kg/m*s)
    rho = 998;            # density (kg/m3)

    F = zeros((6));

    F[0] = f1*L1/D1*rho/2*(4*Q1/pi/D1**2)**2 - \
           f2*L2/D2*rho/2*(4*Q2/pi/D2**2)**2;      # DP_1 - DP_2 = 0
    F[1] = f1*L1/D1*rho/2*(4*Q1/pi/D1**2)**2 - \
           f3*L3/D3*rho/2*(4*Q3/pi/D3**2)**2;      # DP_1 - DP_3 = 0
    F[2] = 1/sqrt(f1)+2*log10(e1/D1/3.7 + \
           2.51/(rho*D1/mu*4*Q1/pi/D1**2/sqrt(f1)));  # Colbrook 1
    F[3] = 1/sqrt(f2)+2*log10(e2/D2/3.7 + \
           2.51/(rho*D2/mu*4*Q2/pi/D2**2/sqrt(f2)));  # Colbrook 2
    F[4] = 1/sqrt(f3)+2*log10(e3/D3/3.7 + \
           2.51/(rho*D3/mu*4*Q3/pi/D3**2/sqrt(f3)));  # Colbrook 3
    F[5] = Q1+Q2+Q3-Qt;                               # total flow

    return F

#--------------------------------------

Qt     = 0.01333;      # Given total volumetric flow rate
xGuess = array([0.01, 0.01, 0.01, 0.004, 0.004, Qt-0.004-0.004]);
                # f1, f2, f3, Q1, Q2, Q3

x      = fsolve(solverF, xGuess)

print "[f1, f2, f3, Q1, Q2, Q3] = ", x
```

Matlab

```matlab
function F=solverF(x)

    f1 = x(1);          % recover variables so eqautions easier to read
    f2 = x(2);
    f3 = x(3);
    Q1 = x(4);
    Q2 = x(5);
    Q3 = x(6);

    Qt = 0.01333;       % Given total volumetric flow rate
    e1 = 0.00024;       % pipe roughness (m)
    e2 = 0.00012;
    e3 = 0.0002;
    L1 = 100;           % pipe length (m)
    L2 = 150;
    L3 = 80;
    D1 = 0.05;          % pipe diameter (m)
    D2 = 0.045;
    D3 = 0.04;
    mu = 1.002E-3;      % viscosity (kg/m*s)
    rho = 998;          % density (kg/m3)

    F(1) = f1*L1/D1*rho/2*(4*Q1/pi/D1^2)^2 - ...
           f2*L2/D2*rho/2*(4*Q2/pi/D2^2)^2;     % DP_1 - DP_2 = 0
    F(2) = f1*L1/D1*rho/2*(4*Q1/pi/D1^2)^2 - ...
           f3*L3/D3*rho/2*(4*Q3/pi/D3^2)^2;     % DP_1 - DP_3 = 0
    F(3) = 1/sqrt(f1)+2*log10(e1/D1/3.7 + ...
           2.51/(rho*D1/mu*4*Q1/pi/D1^2/sqrt(f1))); % Colbrook 1
    F(4) = 1/sqrt(f2)+2*log10(e2/D2/3.7 + ...
           2.51/(rho*D2/mu*4*Q2/pi/D2^2/sqrt(f2))); % Colbrook 2
    F(5) = 1/sqrt(f3)+2*log10(e3/D3/3.7 + ...
           2.51/(rho*D3/mu*4*Q3/pi/D3^2/sqrt(f3))); % Colbrook 3
    F(6) = Q1+Q2+Q3-Qt;                             % total flow

end
```

2 files

```matlab
Qt = 0.01333;        % Given total volumetric flow rate

xGuess = ([0.01 0.01 0.01 0.004 0.004 Qt-0.004-0.004]);
            % f1, f2, f3, Q1, Q2, Q3

x = fsolve(@solverF, xGuess)
```

Mathcad

$Qt := 0.01333$ $\rho := 998$ $\mu := 1.002 \cdot 10^{-3}$

$e1 := 0.00024$ $L1 := 100$ $D1 := 0.05$

$e2 := 0.00012$ $L2 := 150$ $D2 := 0.045$

$e3 := 0.0002$ $L3 := 80$ $D3 := 0.04$

Guesses

$f1 := 0.01$ $Q1 := 0.004$

$f2 := 0.01$ $Q2 := 0.004$

$f3 := 0.01$ $Q3 := Qt - Q1 - Q2$

Given

$$f1 \cdot \frac{L1}{D1} \cdot \frac{\rho}{2} \cdot \left(\frac{4 \cdot Q1}{\pi \cdot D1^2}\right)^2 = f2 \cdot \frac{L2}{D2} \cdot \frac{\rho}{2} \cdot \left(\frac{4 \cdot Q2}{\pi \cdot D2^2}\right)^2$$

$$f1 \cdot \frac{L1}{D1} \cdot \frac{\rho}{2} \cdot \left(\frac{4 \cdot Q1}{\pi \cdot D1^2}\right)^2 = f3 \cdot \frac{L3}{D3} \cdot \frac{\rho}{2} \cdot \left(\frac{4 \cdot Q3}{\pi \cdot D3^2}\right)^2$$

$$\frac{1}{\sqrt{f1}} + 2 \cdot \log\left[\frac{e1}{D1 \cdot 3.7} + \frac{2.51}{\left(\frac{\rho \cdot D1}{\mu} \cdot \frac{4 \cdot Q1}{\pi \cdot D1^2}\right) \cdot \sqrt{f1}}\right] = 0$$

$$\frac{1}{\sqrt{f2}} + 2 \cdot \log\left[\frac{e2}{D2 \cdot 3.7} + \frac{2.51}{\left(\frac{\rho \cdot D2}{\mu} \cdot \frac{4 \cdot Q2}{\pi \cdot D2^2}\right) \cdot \sqrt{f2}}\right] = 0$$

$$\frac{1}{\sqrt{f3}} + 2 \cdot \log\left[\frac{e3}{D3 \cdot 3.7} + \frac{2.51}{\left(\frac{\rho \cdot D3}{\mu} \cdot \frac{4 \cdot Q3}{\pi \cdot D3^2}\right) \cdot \sqrt{f3}}\right] = 0$$

$$Qt = Q1 + Q2 + Q3$$

$$\begin{pmatrix} f1 \\ f2 \\ f3 \\ Q1 \\ Q2 \\ Q3 \end{pmatrix} := \text{Find}(f1, f2, f3, Q1, Q2, Q3) = \begin{pmatrix} 0.031 \\ 0.027 \\ 0.031 \\ 5.775 \times 10^{-3} \\ 3.889 \times 10^{-3} \\ 3.665 \times 10^{-3} \end{pmatrix}$$

Flow through 3 parallel pipes given total flow, pipe props

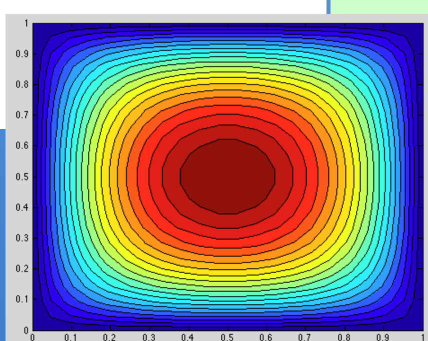
Code: 2D unsteady heat equation

Python

```python
# 2-D unsteady heat equation
# df/dt = alpha*d2f/dx2 + d2f/dy2 + S
# Forward Euler, central difference.
# Finite difference
# Points on boundaries, solve interior points.
# BC = 0; IC = 0

from numpy              import *
from matplotlib.pyplot    import *
from mpl_toolkits.mplot3d import *
from math               import *

Ld       = 1.0                # domain length
nTauRun  = 0.5                # # of diffusion timescales to run
nxy      = 22                 # # of uniform grid points in x, y
alpha    = 1                  # thermal diffusivity
cfl      = 0.5                # time step factor

tau  = Ld**2/alpha            # domain diffusion timescale
tend = nTauRun*tau            # run time
dxy  = Ld/(nxy-1)             # grid spacing
dt   = dxy**2/alpha/4*cfl     # time step size
nt   = ceil(tend/dt)          # number of time steps
dt   = tend/nt                # clean it up
np   = ceil(1/cfl)*10         # how often to plot?

f = zeros((nt,nxy,nxy))       # initialize the solution
S = ones((nt,nxy,nxy))        # set the source term

X,Y = meshgrid(linspace(0,Ld,nxy),linspace(0,Ld,nxy))  # for plotting
i = arange(1,nxy-1)
j = i

for it in arange(1,nt) :

    f[it][ix_(i,j)] = f[it-1][ix_(i,j)] \
                + (alpha*dt/dxy**2)*(f[it-1][ix_(i-1,j)]-2*f[it-1][ix_(i,j)]+f[it-1][ix_(i+1,j)]) \
                + (alpha*dt/dxy**2)*(f[it-1][ix_(i,j-1)]-2*f[it-1][ix_(i,j)]+f[it-1][ix_(i,j+1)]) \
                + S[it-1][ix_(i,j)]

    if(it%np==0) :          # plot
        clf()
        contourf(X,Y,f[it,:,:],20)
        ion(); show()
        aa = raw_input()
```

Matlab

```matlab
% 2-D unsteady heat equation
% df/dt = alpha*d2f/dx2 + d2f/dy2 + S
% Forward Euler, central difference.
% Finite difference
% Points on boundaries, solve interior points.
% BC = 0; IC = 0

Ld       = 1.0;                 % domain length
nTauRun  = 0.5;                 % # of diffusion timescales to run
nxy      = 22;                  % # of uniform grid points in x, y
alpha    = 1;                   % thermal diffusivity
cfl      = 0.5;                 % time step factor

tau  = Ld^2/alpha;              % domain diffusion timescale
tend = nTauRun*tau;             % run time
dxy  = Ld/(nxy-1);              % grid spacing
dt   = dxy^2/alpha/4*cfl;       % time step size
nt   = ceil(tend/dt);           % number of time steps
dt   = tend/nt;                 % clean it up
np   = ceil(1/cfl)*10;          % how often to plot?

f = zeros(nt,nxy,nxy);          % initialize the solution
S = ones(nt,nxy,nxy);           % set the source term

[X,Y] = meshgrid(linspace(0,Ld,nxy),linspace(0,Ld,nxy)); % for plotting
i = 2:nxy-1;
j = i;

for it=2:nt

    f(it,i,j) = f(it-1,i,j) ...
            + (alpha*dt/dxy^2)*(f(it-1,i-1,j)-2*f(it-1,i,j)+f(it-1,i+1,j)) ...
            + (alpha*dt/dxy^2)*(f(it-1,i,j-1)-2*f(it-1,i,j)+f(it-1,i,j+1)) ...
            + S(it-1,i,j);

    if(mod(it,np)==0)           % plot
        Z = reshape(f(it,:,:),nxy,nxy);
        contourf(X,Y,Z,20);
        pause(0.1);
    end

end
```



Finite difference, Euler integration

IPython Notebook

- A wrapper with full Python functionality
- Web interface (and others)
- Workbook
  - Edit and output dynamically
  - Document as you go, with formatting
  - Print/share
  - Embed images and movies, etc.
  - Direct code loading
- Interactive e-books are written in IPython notebook
- Symbolic solver with formatted output

```
In [39]: f=exp(x)+x+a**2

In [40]: integrate(f,x)
```

Out[40]: $a^2 x + \frac{x^2}{2} + e^x$