PHY407 (Computational Physics), University of Toronto

Lecture 1: Python basics

Nicolas Grisouard, nicolas.grisouard@utoronto.ca

13 September 2021

Supporting textbook chapters for week 1: 2, 3 and 4.3

This is an example of "lecture notes". As you will quickly find out, this course is by nature a lab course. Therefore, my "lecture notes" will not often follow the linear progression of regular lecture notes. This is particularly true for this first lecture, in which I merely want to give you pointers to do the first lab.

1 Pseudocode

1.1 General principles

The concept of "Pseudocode" is loosely defined, see for example https://en.wikipedia.org/wiki/Pseudocode. Python is actually seen as pseudocode by some people.

In this lecture, I will use a **very** loose definition for it, i.e., mostly plain English with bullet points. You are free to use your own (and probably better) version. Make sure that it is understandable by someone who speaks English and is familiar with the problem you would be trying to solve, though not familiar with Python.

- Pseudocode is the planned version of your code, written in plain English ($\neq$ programming language).
- You should write one before starting any code.
- It should describe your algorithm.
- It helps ensure that your planned logic for the algorithm is sound.

- Real text or comments for your real code in the end.
- **Keep a copy of it intact** so you can refer to it when you are coding.
- Coding = turning your pseudocode $\rightarrow$ specific programming language. You should be able to take your pseudocode and convert it into any typical programming language.

- Pseudocode: somewhat personal. You do you.
- Concise, logical, step-by-step.
- Start with **brief** overview of what this piece of code will do.

Examples for sequential stuff: * Input: `READ`, `OBTAIN`, `GET` * Initialize: `SET`, `DEFINE` * Compute: `COMPUTE`, `CALCULATE`, `DETERMINE` * Add one: `INCREMENT`, `BUMP` * Output: `PRINT`, `DISPLAY`, `PLOT`, `WRITE`

Examples for conditions and loops: * `WHILE`, `IF-THEN-ELSE`, `REPEAT-UNTIL`, `CASE`, `FOR`

Should also include calling functions: * `CALL`

1.2 Pseudo-code, example 1

Convert polar to Cartesian coordinates from keyboard input:

$r, \theta\ (^\circ) \to x, y$.

Let's try, in code comments form:

[ ]:
```
# Read radius r from keyboard
# Read angle theta in degs from keyboard input
# convert degrees into radians
# Compute x, y = r*(cos(theta), sin(theta))
# print the result to screen
```

[1]:
```
# Read radius from keyboard input
# Read angle in degrees from keyboard input
# Convert angle in radians
# compute x and y from r an theta
# Print the result to screen
```

Alternative: typeset the pseudocode in your report.

[2]:
```
import numpy as np  # import numpy
```

[3]:
```
# From keyboard, read the radius and save.
r = float(input("Enter r: "))
```

```
Enter r: 13
```

[4]:
```
# From keyboard, read the angle in degrees and save.
theta = float(input("Enter theta in degs: "))
```

```
Enter theta in degs: 45
```

[7]:
```
# Do the conversion from degrees to radians
theta_r = theta/180*np.pi
```

[8]:
```
# Compute $(x, y) = r(\cos\theta_r, \sin\theta_r)$.
x, y = r*np.cos(theta_r), r*np.sin(theta_r)
```

[10]:
```
# Print result (x, y) to screen.
print("x = {0:.4f}, y = {1:.2e}".format(x, y))
```

```
x = 9.1924, y = 9.19e+00
```

1.3 Pseudo-code, example 2

- Number of radioactive atoms of uranium N as a function of time given initial N_0 and
$$\frac{dN}{dt} = -\frac{N}{\tau}.$$
(τ = decay constant)
- Use Euler method for integration.
- Integrate for 5τ.

[11]:
```
# Pseudocode:
# Define decay constant tau, initial N0 and duration 5tau
# Initialize time and time array, as well as N(t) array
# Compute dt
# FOR 100 iterations:
#     Increment the N array from the previous values according to Euler
# Plot N vs. t
```

[12]:
```
""" Code that computes the number of radioactive atoms of uranium N
as fct of time, given tau and N0 and using Euler method.
Author: Nicolas Grisouard, Pythonisto extraordinaire, Sept. 2021 """

# Re-import, in case I am executing cells in random order
import numpy as np
import matplotlib.pyplot as plt
```

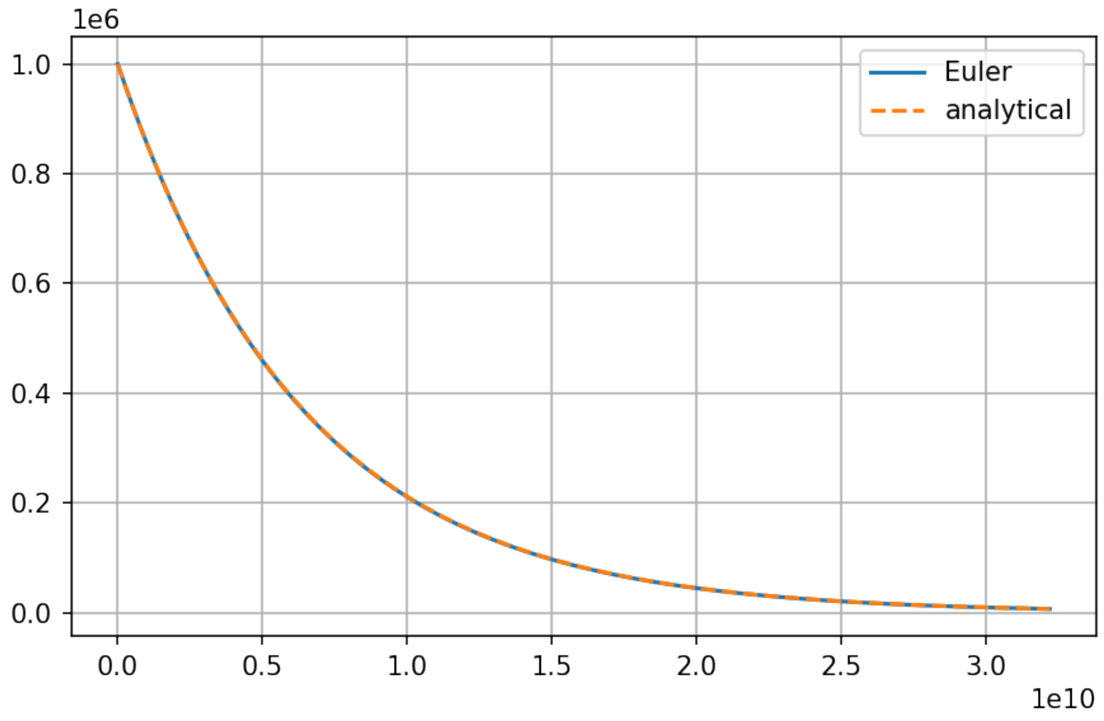
[ ]:
```
# I skipped this cell in class: it is a conversion of the
# half-life of U238 (which is easily found on Wikipedia)
# and the decay rate (which for some reason is not)
print("{0:1.3e}".format(4.468e9/np.log(2.)))
```

[13]:
```
# 1. Define decay constant tau, initial N0, and end time 5*tau
tau = 6.446e9   # [s] tau for U238
N0 = 1e6   # [1] initial N
t_end = 5*tau   # [s] the end of the simulation time
```

[24]:
```
# 2. Initialize time array with 101 values
time = np.linspace(0., t_end, 1001)

# 3. Compute dt the time step because we'll need it
dt = time[1] - time[0]

# 4. Initialize a number array with same number of elements
N = 0*time
N[0] = N0
```

[25]:
```
# 5. FOR 101 iterations:
#         Increment the N array with Euler: N[i+1] = N[i] - dt*N[i]/tau
for ii in range(len(time)-1):
    N[ii+1] = N[ii] - dt*N[ii]/tau
```

[26]:
```
# 6. Plot N vs. t
plt.figure(dpi=150)
plt.plot(time, N, label='Euler')
plt.plot(time, N0*np.exp(-time/tau), '--', label='analytical')
plt.grid()
plt.legend()
plt.tight_layout()
```



2 Typical approach for solving a problem

1. Start with math model, often but not always continuous.
2. **Discretize:** set up discrete arrays of independent variables (e.g., x, t), dependent variables (e.g. $v(t)$, $a(t)$), and define operators on these variables (dv/dt, $ma\ldots$).
3. **Initialize** parameters and variables appropriately.
4. **Evalutize:** run algorithm to operate on these variables.
5. **Analyze:** some extra processing of the raw results, figures…

See example: Given position $x(t)$ of a particle undergoing SHO with angular frequecy 0.5 rad/s, initial position 3 m, initial velocity 0 m/s, calculate velocity and acceleration using simple finite difference

```
[27]: # Example to illustrate general procedure for mathematical modeling.
      # Given position x(t) of a particle undergoing SHO,
      # calculate velocity and acceleration using simple finite difference
      import numpy as np  # import numpy
      import matplotlib.pyplot as plt  # import figure functions
```

```
[37]: # 2. Discretize

      # Define time grid and dependent variables
      omega = 0.5  # [rad/s] angular frequency
      A = 3.  # [m] diplacement amplitude
      t = np.linspace(0, 30., 301)  # time array: linear space from 0 to 10s, 101␣
       ↪elements
      N = len(t)  # number of time steps (length of time array)
      x = np.empty(N)  # array of positions (empty array of length N)
      v = np.empty(N-1)  # array of velocities (empty array of length N-1)
      a = np.empty(N-2)  # array of accelerations (empty array of length N-2)
```

```
[38]: # 3. Initialize: define signal on discretized grid
      x = A*np.cos(omega*t)
```

```
[39]: # 2. discretize and 4. evalutize (here, apply algorithm)
      # Define velocity using finite differences: v = Delta x/Delta t
      dt = t[1] - t[0]
      for k in range(N-1):
          v[k] = (x[k+1]-x[k])/dt

      # Define acceleration using finite differences: a = Delta v/Delta t
      for k in range(len(x)-2):
          a[k] = (v[k+1]-v[k])/dt
```

```
[40]: # 5. Analyze
      # print results
      print("t is ", t)
```

```
t is  [ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3
  1.4  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7
  2.8  2.9  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1
  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5
  5.6  5.7  5.8  5.9  6.   6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
  7.   7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.   8.1  8.2  8.3
  8.4  8.5  8.6  8.7  8.8  8.9  9.   9.1  9.2  9.3  9.4  9.5  9.6  9.7
  9.8  9.9 10.  10.1 10.2 10.3 10.4 10.5 10.6 10.7 10.8 10.9 11.  11.1
 11.2 11.3 11.4 11.5 11.6 11.7 11.8 11.9 12.  12.1 12.2 12.3 12.4 12.5
 12.6 12.7 12.8 12.9 13.  13.1 13.2 13.3 13.4 13.5 13.6 13.7 13.8 13.9
 14.  14.1 14.2 14.3 14.4 14.5 14.6 14.7 14.8 14.9 15.  15.1 15.2 15.3
 15.4 15.5 15.6 15.7 15.8 15.9 16.  16.1 16.2 16.3 16.4 16.5 16.6 16.7
 16.8 16.9 17.  17.1 17.2 17.3 17.4 17.5 17.6 17.7 17.8 17.9 18.  18.1
 18.2 18.3 18.4 18.5 18.6 18.7 18.8 18.9 19.  19.1 19.2 19.3 19.4 19.5
 19.6 19.7 19.8 19.9 20.  20.1 20.2 20.3 20.4 20.5 20.6 20.7 20.8 20.9
 21.  21.1 21.2 21.3 21.4 21.5 21.6 21.7 21.8 21.9 22.  22.1 22.2 22.3
 22.4 22.5 22.6 22.7 22.8 22.9 23.  23.1 23.2 23.3 23.4 23.5 23.6 23.7
 23.8 23.9 24.  24.1 24.2 24.3 24.4 24.5 24.6 24.7 24.8 24.9 25.  25.1
 25.2 25.3 25.4 25.5 25.6 25.7 25.8 25.9 26.  26.1 26.2 26.3 26.4 26.5
 26.6 26.7 26.8 26.9 27.  27.1 27.2 27.3 27.4 27.5 27.6 27.7 27.8 27.9
 28.  28.1 28.2 28.3 28.4 28.5 28.6 28.7 28.8 28.9 29.  29.1 29.2 29.3
 29.4 29.5 29.6 29.7 29.8 29.9 30. ]
```

[41]:
```
print("x is ", x)
```

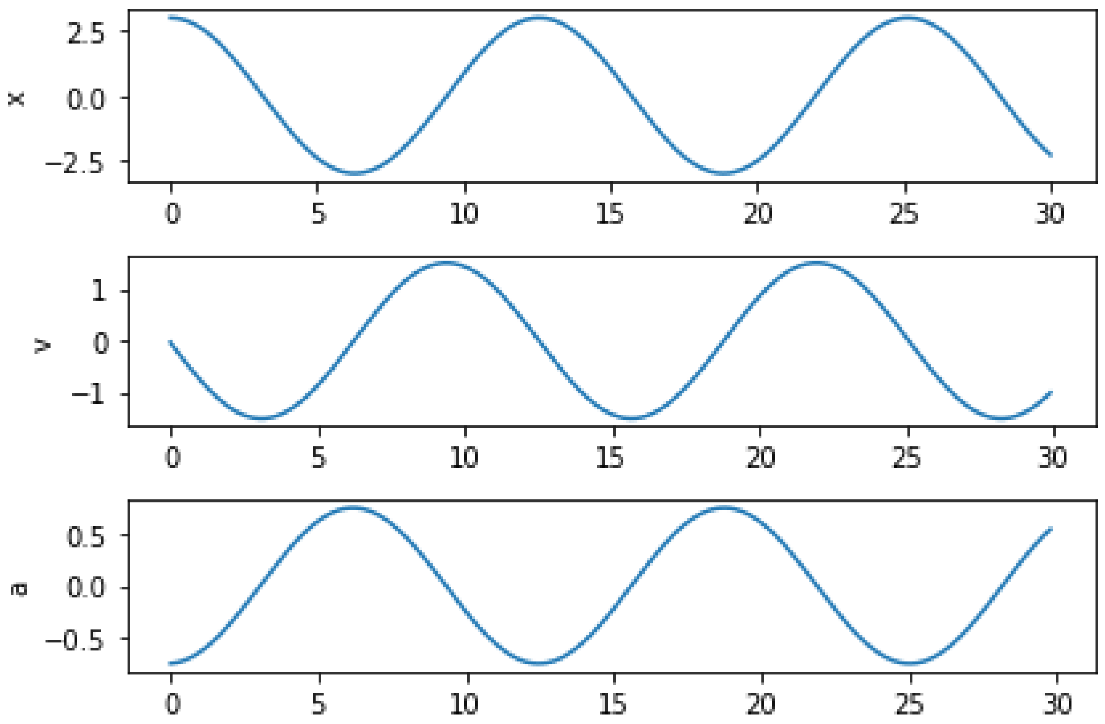
```
x is  [ 3.          2.99625078  2.9850125   2.96631323  2.94019973  2.90673727
  2.86600947  2.81811814  2.76318298  2.70134131  2.63274769  2.55757357
  2.47600684  2.3882514   2.29452656  2.19506661  2.09012013  1.97994944
  1.8648299   1.74504927  1.62090692  1.49271314  1.36078836  1.22546232
  1.08707326  0.94596709  0.80249649  0.65702006  0.50990143  0.36150831
  0.21221161  0.06238448 -0.08759857 -0.23736267 -0.38653348 -0.53473817
 -0.68160628 -0.82677074 -0.9698687  -1.11054249 -1.24844051 -1.38321807
 -1.51453831 -1.642073   -1.76550335 -1.88452087 -1.99882806 -2.10813923
 -2.21218115 -2.31069376 -2.40343085 -2.49016061 -2.57066626 -2.64474659
 -2.71221643 -2.77290714 -2.82666702 -2.87336171 -2.9128745  -2.94510661
 -2.96997749 -2.98742497 -2.99740545 -2.99989398 -2.99488433 -2.98238903
 -2.96243931 -2.93508504 -2.90039458 -2.85845464 -2.80937006 -2.75326352
 -2.69027525 -2.62056269 -2.5443001  -2.46167807 -2.37290314 -2.27819718
 -2.17779691 -2.07195329 -1.96093086 -1.84500713 -1.72447184 -1.59962627
 -1.47078246 -1.33826247 -1.20239752 -1.0635272  -0.92199861 -0.77816551
 -0.6323874  -0.48502865 -0.33645758 -0.18704554 -0.03716599  0.11280646
  0.26249695  0.41153134  0.55953711  0.70614433  0.85098656  0.99370176
  1.13393323  1.27133045  1.40555001  1.53625643  1.66312301  1.78583264
  1.90407863  2.01756541  2.12600932  2.22913932  2.32669764  2.41844042
  2.50413835  2.58357725  2.65655855  2.72289984  2.78243529  2.83501611
  2.88051086  2.91880583  2.94980532  2.97343182  2.98962629  2.99834825
  2.99957591  2.99330619  2.97955476  2.95835599  2.92976288  2.89384688
  2.85069778  2.80042341  2.74314944  2.67901903  2.60819247  2.53084679
  2.4471753   2.35738715  2.26170676  2.16037329  2.05364     1.94177368
  1.82505394  1.70377252  1.57823255  1.44874782  1.31564198  1.17924772
  1.03990595  0.89796496  0.75377953  0.60771004  0.46012159  0.31138307
  0.16186626  0.01194487 -0.13800638 -0.28761268 -0.4365001  -0.5842965
 -0.73063246 -0.87514222 -1.01746458 -1.15724381 -1.29413053 -1.42778261
 -1.55786596 -1.68405547 -1.80603571 -1.9235018  -2.03616014 -2.14372914
 -2.24593994 -2.34253705 -2.43327904 -2.5179391  -2.59630563 -2.66818274
 -2.73339079 -2.79176678 -2.84316481 -2.8874564  -2.92453086 -2.95429552
 -2.97667598 -2.99161629 -2.99907913 -2.99904582 -2.99151647 -2.97650988
 -2.95406357 -2.92423363 -2.88709464 -2.84273941 -2.79127882 -2.73284148
 -2.66757346 -2.59563789 -2.51721459 -2.43249955 -2.34170454 -2.24505648
 -2.14279696 -2.03518155 -1.92247925 -1.80497176 -1.68295277 -1.55672728
 -1.42661078 -1.2929285  -1.15601457 -1.01621121 -0.87386785 -0.72934027
 -0.58298972 -0.435182   -0.28628655 -0.13667554  0.01327709  0.16319654
  0.31270808  0.46143801  0.60901459  0.75506895  0.89923603  1.04115549
  1.1804726   1.31683914  1.44991428  1.57936538  1.70486889  1.82611112
  1.94278902  2.05461096  2.16129744  2.2625818   2.35821089  2.44794568
  2.53156188  2.60885049  2.67961833  2.74368853  2.80090093  2.85111254
  2.89419785  2.93004918  2.95857691  2.97970973  2.99339484  2.99959801
  2.99830376  2.9895153   2.97325462  2.94956234  2.9184977   2.88013833
  2.83458011  2.78193693  2.72234034  2.65593933  2.58289985  2.50340446
  2.41765187  2.32585641  2.22824752  2.12506917  2.01657925  1.90304894
  1.78476199  1.66201407  1.53511198  1.40437291  1.27012363  1.13269971
  0.99244463  0.84970896  0.70484946  0.5582282   0.41021165  0.2611698
  0.11147515 -0.03849812 -0.18837517 -0.33778138 -0.48634331 -0.63368964
 -0.77945207 -0.92326628 -1.0647728  -1.20361795 -1.33945467 -1.47194346
 -1.60075316 -1.72556181 -1.84605745 -1.96193891 -2.07291654 -2.17871296
 -2.27906374]
```

[42]:
```
print("a is ", a)
```

```
a is  [-0.74890665 -0.74609767 -0.74142383 -0.73489681 -0.72653294 -0.71635311
 -0.70438277 -0.69065184 -0.67519464 -0.65804981 -0.6392602  -0.61887276
 -0.59693847 -0.57351214 -0.54865233 -0.52242118 -0.49488425 -0.46611036
 -0.43617144 -0.40514231 -0.37310055 -0.34012622 -0.30630176 -0.2717117
 -0.23644251 -0.20058233 -0.1642208  -0.1274488  -0.09035825 -0.05304185
 -0.01559287  0.02189508  0.0593283   0.09661324  0.13365669  0.17036607
  0.20664963  0.24241667  0.27757779  0.31204511  0.34573248  0.3785557
  0.41043273  0.44128389  0.47103207  0.49960292  0.52692502  0.55293008
  0.5775531   0.60073254  0.62241047  0.64253269  0.66104891  0.67791286
  0.69308237  0.70651955  0.71819079  0.72806692  0.73612327  0.7423397
  0.74670066  0.74919526  0.74981726  0.74856511  0.74544194  0.74045555
  0.7336184   0.72494759  0.7144648   0.70219621  0.68817249  0.67242871
  0.6550042   0.63594252  0.61529132  0.59310221  0.56943065  0.54433581
  0.51788042  0.49013059  0.4611557   0.43102815  0.39982326  0.36761902
  0.33449592  0.30053676  0.26582641  0.23045164  0.19450085  0.15806392
  0.1212319   0.08409687  0.04675164  0.00928956 -0.02819574 -0.06561057
 -0.1028614  -0.13985514 -0.17649931 -0.21270232 -0.24837369 -0.28342425
 -0.3177664  -0.3513143  -0.3839841  -0.41569414 -0.44636516 -0.47592049
 -0.50428628 -0.53139161 -0.55716874 -0.58155324 -0.60448415 -0.62590418
 -0.64575976 -0.66400129 -0.68058315 -0.69546392 -0.70860638 -0.7199777
 -0.72954945 -0.73729771 -0.7432031  -0.74725088 -0.74943091 -0.74973776
 -0.74817066 -0.74473352 -0.73943493 -0.73228814 -0.72331101 -0.71252598
 -0.69996001 -0.6856445  -0.66961524 -0.65191229 -0.63257989 -0.61166638
 -0.58922402 -0.5653089  -0.53998081 -0.51330305 -0.4853423  -0.45616844
 -0.4258544  -0.39447595 -0.36211151 -0.32884198 -0.29475052 -0.25992233
 -0.22444448 -0.18840563 -0.15189586 -0.11500643 -0.07782955 -0.04045814
 -0.0029856   0.03449441  0.07188819  0.10910229  0.1460437   0.18262006
  0.21873998  0.25431316  0.28925068  0.32346524  0.35687129  0.38938536
  0.42092616  0.45141487  0.48077528  0.50893399  0.53582064  0.56136802
  0.58551227  0.60819304  0.62935364  0.64894119  0.66690673  0.68320534
  0.6977963   0.71064313  0.72171373  0.73098041  0.73842002  0.74401397
  0.74774827  0.74961359  0.74960527  0.74772332  0.74397246  0.73836205
  0.73090612  0.7216233   0.71053681  0.69767434  0.68306805  0.66675444
  0.6487743   0.62917255  0.60799821  0.58530418  0.5611472   0.53558764
  0.5086894   0.48051969  0.45114894  0.42065055  0.38910075  0.3565784
  0.32316479  0.28894344  0.25399988  0.21842145  0.18229708  0.14571707
  0.10877284  0.07155673  0.03416177 -0.00331858 -0.04079064 -0.07816073
 -0.11533547 -0.15222193 -0.18872791 -0.22476218 -0.26023465 -0.29505667
 -0.32914121 -0.36240306 -0.39475909 -0.42612843 -0.45643268 -0.48559608
 -0.51354574 -0.5402118  -0.56552762 -0.58942991 -0.61185893 -0.63275863
 -0.65207676 -0.66976503 -0.68577924 -0.70007936 -0.71262965 -0.72339874
 -0.7323597  -0.73949015 -0.74477225 -0.74819282 -0.74974329 -0.74941979
 -0.74722313 -0.74315881 -0.73723698 -0.72947243 -0.71988459 -0.70849741
 -0.69533935 -0.68044331 -0.66384651 -0.64559045 -0.62572074 -0.60428706
 -0.58134297 -0.55694583 -0.53115662 -0.50403979 -0.47566313 -0.44609755
 -0.41541696 -0.38369805 -0.35102009 -0.31746476 -0.28311594 -0.24805947
 -0.21238299 -0.17617566 -0.13952798 -0.10253155 -0.06527885 -0.02786298
  0.00962253  0.04708398  0.08442775  0.1215605   0.15838941  0.19482242
  0.23076849  0.26613775  0.3008418   0.33479391  0.36790921  0.40010492
  0.43130059  0.46141822  0.49038255  0.51812118  0.54456478]
```

[43]:
```python
# plot results
def plot_da_fig(t, x, v, a, N):
    plt.figure()

    plt.subplot(3, 1, 1)
    plt.plot(t, x)
    plt.ylabel('x')

    plt.subplot(3, 1, 2)
    plt.plot(t[:-1], v)
    plt.ylabel('v')

    plt.subplot(3, 1, 3)
    plt.plot(t[:-2], a)
    plt.ylabel('a')

    plt.tight_layout()
    # plt.savefig('T01.pdf')  # saves a pdf figure on disk
```

[44]:
```python
plot_da_fig(t, x, v, a, N)
```


x
2.5
0.0
−2.5
v
1
0
−1
a
0.5
0.0
−0.5
0
5
10
15
20
25
30