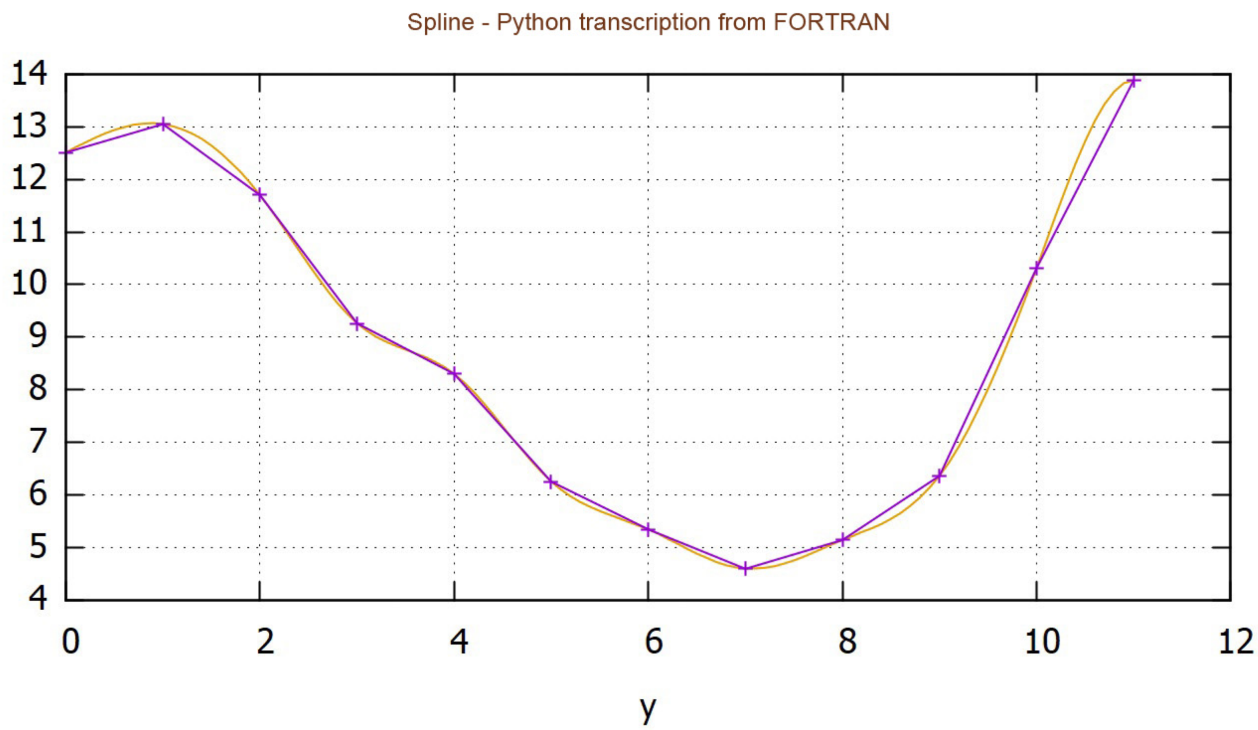
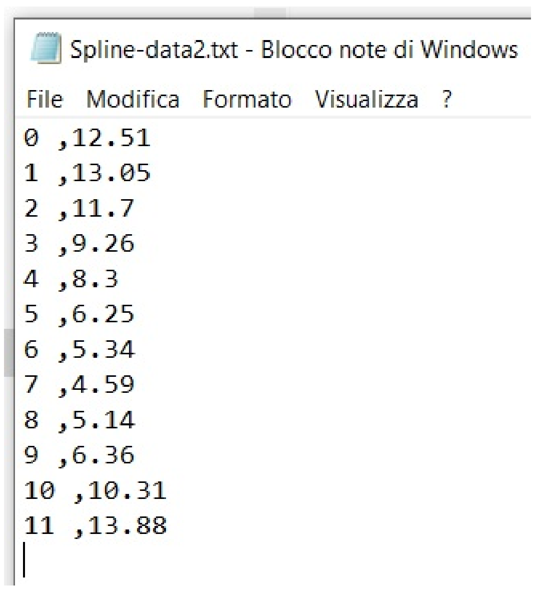

Spline - Python transcription from FORTRAN
14
13
12
11
10
9
8
7
6
5
4
0
2
4
6
8
10
12
y

```python
# -*- coding: utf-8 -*-

# Created on Sun 2021-10-31
# Source material = FORTRAN CODE from Web
# Python transcription = Giampiero Barbieri

# spline-p.py --- cubic spline for a tabulated function


def splint(xa, ya, y2a, n, x):
      import sys
      #
      # INPUT:       xa[i], ya[i] i = 0, n-1 = tabulated function
      #                x1 <x2 <... <xN
      #              y2a[i]       i = 0, n-1 = second derivatives
      #              x = generic value with:   xa[0] <= x <= xa[n-1]
      #
      # OUTPUT       y = cubic-spline interpolated value
      #
      # INTEGER k, khi, klo
      # REAL a, b, h
      #
      # We find the right place in the table by means of bisection.
      # This is optimal if sequential calls to this routine are at
      # random values of x.
      #
      # If sequential calls are in order, and closely
      # spaced, one would do better to store previous values of
      # klo and khi and test if they remain appropriate on the
      # next call.
      #
      klo = 0
      khi = int(n - 1)
      #
      while khi-klo > 1:
          k = int((khi + klo) / 2)
          if xa[k] > x:
              khi = k
          else:
              klo = k
      #
      # klo and khi now bracket the input value of x.
      h = xa[khi] - xa[klo]
      if h == 0:
          sys.exit()
      #
      # we have bad xa input in splint:

      a = (xa[khi] - x) / h
      #
      # Cubic spline polynomial is now evaluated.
      b = (x - xa[klo]) / h
      #
      y = a * ya[klo] + b * ya[khi] + ((a**3 - a) * y2a[klo]
          + (b**3 - b) * y2a[khi]) * (h**2) / 6.
      return y
      #
      # Cubic spline polynomial is now evaluated.
```

```
#
# **************************************************************************
def spline(x,y,n,yp1,ypn,y2,NMAX=500):
      # INPUT:
      # n    = number of samples
      # NMAX = maximum number admissible for n
      # yp1  = value of the derivative in the first point - default = 0
      # ypn  = value of the derivative in the last  point - default = 0
      # x    = array of values = independent variable
      # y    = array of values = tabulated function
      #
      # OUTPUT:
      # y2   = array of values = second derivatives of the function
      #        at the tabulated points x
      #
      #        If yp1 and/or ypn >= 0.99e30 then
      #        the routine sets the corresponding boundary
      #        condition for a natural spline, with
      #        zero second derivative on that boundary.
      #
      # INTEGER i, NMAX
      # REAL un, qn, sig, un, p
      # REAL aa, bb, sig, p, cc, dd, ee, ff, zz = auxiliary variables
      # ARRAY u[n], y2[n]

      # we initializw the array u
      u = np.ones(n+1,dtype=float)
      if (yp1>.99e30):
          #  The lower boundary condition is set either to be “natural”
          y2[0]=0.0
          u[0]=0.0
      else:
          # Using a specified first derivative.
          y2[0]=-0.5
          aa=x[1]-x[0]
          bb=y[1]-y[0]
          cc=yp1
          dd=x[0]
          zz=3.0/aa
          zz=zz*bb
          zz=zz/aa
          zz=zz-cc
          u[0]=zz

      for i in range(1,n-1):
          # This is the decomposition loop of the tridiagonal
          # algorithm. y2 and u are used for temporary
          # storage of the decomposed factors.
          sig=(x[i]-x[i-1])/(x[i+1]-x[i-1])
          p=sig*y2[i-1]+2.
          y2[i]=(sig-1.)/p
          aa=y[i+1]-y[i]
          bb=x[i+1]-x[i]
          cc=y[i]-y[i-1]
          dd=x[i]-x[i-1]
          ee=x[i+1]-x[i-1]
          ff=u[i-1]
          zz=(6.*(aa/bb-cc/dd)/ee-sig*ff)/p
          u[i]=zz
```

```
        if (ypn>.99e30):
            # The upper boundary condition is set either to be “natural”
            qn = 0.
            un = 0.
        else:
            # Using a specified first derivative.
            qn = 0.5
            aa = x[n-1]-x[n-2]
            bb = y[n-1]-y[n-2]
            cc = x[n-1]-x[n-2]
            un = (3./(aa)) * (ypn-(bb) / (cc))

        aa = un-qn*u[n-2]
        y2[n-1]=(un-qn*u[n-2])/(qn*y2[n-2]+1.)


        for k in range(n-2,-1,-1):
            # This is the backsubstitution loop of the tridiagonal algorithm.
            y2[k]=y2[k]*y2[k+1]+u[k]

# ************************************************************************
#
# MAIN PROGRAM
#

# INTEGER n, i, icount, NMAX
# REAL yp1, ypn, xx1, yy1
# ARRAY x(n),y(n),y2(n)
# CHARACTER*64 dirpath, nofi, nofiw, linea, xx
#

import os
import numpy as np
import matplotlib.pyplot as plt

#
# find current dir -- nofi = input data file
dir_path = os.path.dirname(os.path.realpath(__file__))
nofi = dir_path + '/Spline-data2.txt'

# we find the number of line inside nofi
n = -1
with open(nofi, 'r') as f:
    for linea in f:
        n += 1
        xx = linea
f.close()

NMAX = 500
# we initialize the three arrays
x = np.ones(n+1,dtype=float)
y = np.ones(n+1,dtype=float)
y2 = np.ones(n+1,dtype=float)
```



Spline-data2.txt - Blocco note di Windows

File Modifica Formato Visualizza ?

```
0 ,12.51
1 ,13.05
2 ,11.7
3 ,9.26
4 ,8.3
5 ,6.25
6 ,5.34
7 ,4.59
8 ,5.14
9 ,6.36
10 ,10.31
11 ,13.88
```

```python

# we read each single line in nofi
# and we put the strings insides x[n], y[n]
i = -1
with open(nofi, 'r') as f:
    for linea in f:
        i += 1
        xx = linea
        result = xx.find(',')
        y[i] = float(xx[result+1:])
        x[i] = float(xx[0:result])
f.close()

# we find the number of line inside nofi
n = len(x)
# we I define the derivative at the ends of the interval -- [default = 0]
yp1 = 0.0
ypn = 0.0

# we compute the derivative of the function in each point of the
# corresponding array [ y2[n] ]
spline(x, y, n, yp1, ypn, y2, NMAX=NMAX)


# we compute the number of the sample of the spline --- the could be evalueted
# in each point of the definition interval
icount = 0
xx1 = -0.1
yy1 = 1.0
xx = []
yy = []
for i in range(0,10*n-9-1):
      icount = icount + 1
      xx1 = xx1 + 0.1
      xx1 = int(xx1*100+0.5)/100
      yy1 = splint(x,y,y2,n,xx1)
      yy1 = int(yy1*100+0.5)/100
      xx.append(xx1)
      yy.append(yy1)

# Configure the contour
plt.title("Spline interpolation")
plt.plot(x, y)
plt.plot(xx, yy)



# Show the result in the plot window
plt.show()


```