Sakari Cajanus
82036R
sakari.cajanus@aalto.fi

Exercise Round 2

S-114.1100 Computational Science

September 25, 2011

Problem 1. Plots and conclusions

In exercise round 2 we studied polynomial interpolation. In problem 4, we tried to approximate the *serpentine curve*

$$f(x) = \frac{x}{1/4 + x^2}$$

using 13 equidistant nodes and again using 13 *Chebyshev nodes* which were calculated using

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left[\left(\frac{i}{n}\right)\pi\right] \qquad (0 \le i \le n).$$

The interval $[a, b]$ used in both cases was $[-2.02857, 2.02857]$. In figure 1 is shown our function itself and the two approximations. As predicted, the approximation done using *Chebyshev nodes* is much better near the ends of our interval. However, as is shown in

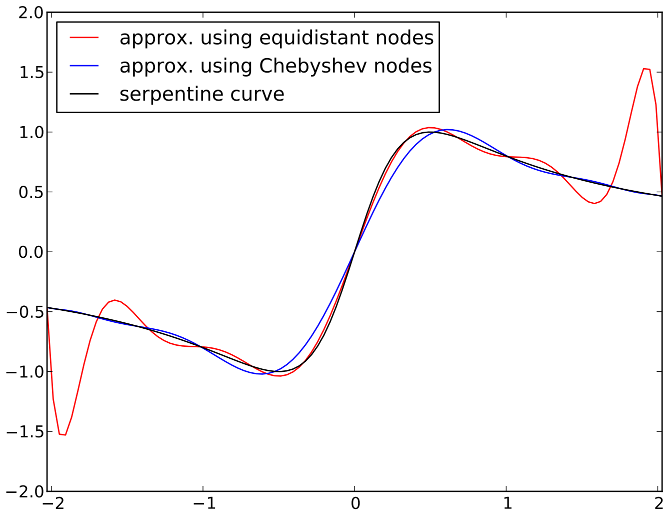

Figure 1: The serpentine curve and polynomial approximations

figure 2 which shows us the errors $|p(x) - f(x)|$ calculated in 101 equidistant points in our interval, the error using *Chebyshev nodes* is actually larger near zero. We can not be satisfied with the results, as better ways can be found to approximate this curve.

We usually choose to use *Chebyshev nodes*, because they help to minimize the *Runge's phenomenon*: problem of oscillation at the edges of an interval when using polynomials of high degree. In the case of *serpentine curve*, approximation near the middle of the interval is also hard.

As seen in figure 1, shape of the *serpentine curve* is hard to approximate, even when using a polynomial of 12th degree. Better approximation might be possible if the points were chosen by hand. In this case, however, a good approximation could be done using natural cubic splines.

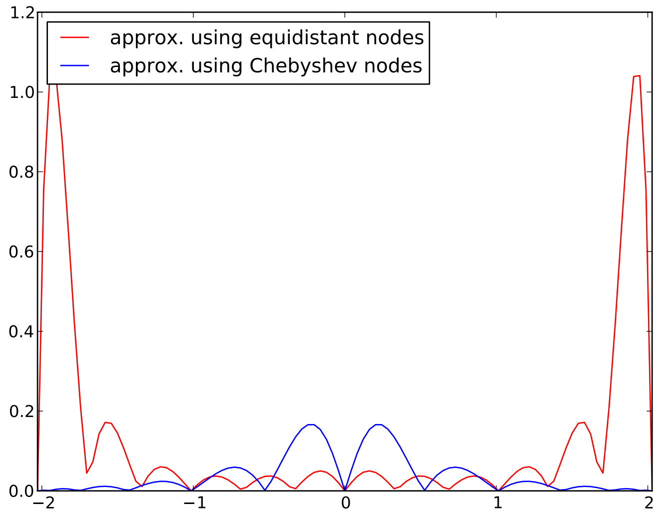

Figure 2: Errors of the polynomial approximations

Appendix 1. Code

```
from __future__ import division
from pylab import *

def coefficients(n, x, y, a):
    a=y;
    for j in arange(1,n+1):
        print j
        for i in linspace(n,j,n-j+1):
            a[i] = (a[i]-a[i-1])/(x[i]-x[i-j])
    return a

def eval(n, x, t, a):
    pt = a[n]
    for i in linspace(n-1,0,n):
        pt = pt * (t-x[i])+a[i];
    return pt

def f(x):
    return x/(1/4.0 + x**2)

def main():
    col = ['r', 'b']
    for k in range(2):
        chebysnev = k
        if(chebysnev==0):
            x = linspace(-2.02857, 2.02857, 13)
        else:
            i = linspace(0,12,13)
            x = 2.02857*cos(pi*i/12)

        print "Hello!"
        x_101 = linspace(-2.02857, 2.02857, 101)

        y = f(x)
        y_101 = f(x_101)
        a = y
        n = size(x)-1

        a = coefficients(n,x,y,a)

        y_eval = linspace(0,0,101)

```

```
        for i in linspace(0,100,101):
            y_eval[i] = eval(n,x,x_101[i],a)
        plot(x_101,abs(y_eval-y_101), col[k])
        #plot(x_101,y_eval, col[k])
    #plot(x_101, y_101, ’k’)
    xlim(min(x_101),max(x_101))
    #legend((’approx. using equidistant nodes’,’approx. using Chebyshev \
#nodes’,’serpentine curve’),loc=2)
    legend((’approx.␣using␣equidistant␣nodes’,’approx.␣using␣Chebyshev␣\
nodes’),loc=2)
    show()

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