Innlevering 4

Innleveringsfrist: fredag 17.04.2026 kl 16:00.

Du leverer besvarelsen din som to filer:

Oppgave 1: Finnes her. Legg til svarene dine og lever Jupyter notebook gjennom Ovsys2. [teller 25%]

Oppgaveene 2-7: Finnes i Oppgave filen på wikisiden og besvarelsen leveres som en pdf fil gjennom Ovsys2. [teller 75%]

Innleveringer levert i feil format underkjennes.

Polynomial interpolation, piecewise-polynomial interpolation and adaptive quadrature.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
# functions

def f1(x):
    y=1/(1+x**2)
    return y

def f2(x):
    y=np.cos(2*np.pi*x)
    return y

def f3(x):
    y=np.exp(3*x)*np.sin(2*x)
    return y

def interpolate(x, x_values, y_values):
    def _basis(j):
        p=1
        for m in range(0,k):
            if m !=j:
                p = p*(x - x_values[m])/(x_values[j] - x_values[m])
        return p
    assert len(x_values) != 0 and (len(x_values) == len(y_values)), 'x and y cannot be empty and must have the same length'
    k = len(x_values)
    return sum(_basis(j)*y_values[j] for j in range(k))

def phi(r, ep):
    #ep = 1.
    return np.exp(-(ep*r)**2)
```

The Python function interpolate(, ,) implements the Lagrange interpolation polynomial for an arbitary set of distinct nodes $x_0, \ldots, x_n$ and for values $y_0, \ldots, y_n$. In what follows we test this code on equidistant nodes and on Chebishev nodes. The code works both for functions $f \in C^\infty$ defined on the interval $[-1, 1]$ and for arbitrary intervals $[a, b]$ by applying the one-to-one transformation $\Psi : [-1, 1] \to [a, b]$, $\Psi(x) = \frac{b-a}{2}x + \frac{b+a}{2}$.

interpolate(, ,) receives in input:

- The data that we want to interpolate: x_i and y_i, $i = 0, \ldots, n$.
- The value of x where we want to evaluate the interpolation polynomial, x can be an array.

and gives in output: the value of the interpolation polynomial in x.

In what follows you can find a plot of the interpolation of the Runge function $f(x) = 1/(x^2 + 1)$ with equidistant nodes and Chebishev nodes on the interval $[-5, 5]$ with $n = 10$.

For obtaining a nice plot, we evaluate the interpolation polynomial on a finer grid denoted here with "xfine".

Familiarize yourself with this code which you will have to use for answering the questions in this notebook.

In [ ]:
```
# Interpolation of the Runge function on [-5,5] with equidistant and Chebishev nodes
# n=10
#
#Interval of approximation
a=-5
b=5


n=10
# Equidistant nodes
x=np.linspace(a, b, num=n+1)
y=f1(x)

# Chebishev nodes
xcheb=np.zeros(n+1)
for j in range(n+1):
    xcheb[j]=np.cos((j+1/2)/(n+1))*np.pi)

#transforming the Chebishev nodes to a generic interval [a,b]

abxcheb=(b-a)/2*xcheb+(b+a)/2
ycheb=f1(abxcheb)


xfine=np.linspace(a, b, num=len(x)*10)
yfine=f1(xfine)
# Evaluating the interpolation polynomial on equidistant nodes on a finer grid
yfine=interpolate(xfine,x,y)
# Evaluating the interpolation polynomial on Chebishev nodes on a finer grid
ychebfine=interpolate(xfine,abxcheb,ycheb)
# Plotting the results
plt.plot(xfine, f1(xfine),'k',
         xfine,yfine,'g',
         x, y,'go',
         xfine, ychebfine,'r',
         abxcheb,ycheb,'ro')
```

a) The interpolation on equidistant nodes and on Chebishev nodes should converge for both the functions:

- $f(x) = \cos(2\pi x)$, $x \in [0, 1]$;
- $f(x) = e^{3x} \sin(2x)$, $x \in [0, \pi/4]$.

Can you use the theory learned in class and explain why? These functions are among those implemented in the preamble.

Explain here why we should get convergence with these two functions and on these two intervals. ...

b)To check convergence in a numerical experiment consider an approximation of

$$\max_{x \in [a,b]} |f(x) - p_n(x)|,$$

and consider the behaviour as n increases.

Consider a grid which has many more points than the maximal number of interpolation nodes: $x_0 = \eta_0 < \eta_1 < \cdots < \eta_N = x_n$ with N large", e.g. $N = 10 n_{\max}$, where $n_{\max}$ is the largest degree of the interpolation polynomial that you consider in your numerical experiments. Compute the following approximations

$$\max_{x \in [a,b]} |f(x) - p_n(x)| \approx \max_{\eta_0, \cdots, \eta_N} |f(\eta_i) - p_n(\eta_i)|.$$

Make a plot of the estimated error $\max_{x \in [a,b]} |p_n(x) - f(x)|$ as a function of n to check convergence. Use semi-logarithmic plot (with logarithmic scale on the y-axis): "plt.semilogy".

Please complete the code below.

In [ ]:
```
#
# Interpolation of f2 on [0,1] and of f3 on [0,pi/4]
#
# n=10
#
#Interval of approximation
a=0
b=1


# Total number of experiments
nmax=20

# Making the fine grid

xfine=np.linspace(a, b, num=len(x)*10)
yfine=f2(xfine)


errmax=np.zeros(nmax)
errmaxCh=np.zeros(nmax)
for k in range(nmax):
    # Make the grid with equidistant nodes
    x=
    # Evaluate f2 on the grid
    y=f2(x)
    # Make the grid with Chebishev nodes on [-1,1]
    xcheb=np.zeros(k+1)
    for j in range(k+1):
        xcheb[j]=

    # Transform the Chebishev nodes to the isterval [a,b]
    abxcheb=
    # Evaluate f2 on the Chebisev nodes
    ycheb=f2(abxcheb)
    # Interpolate on the equidistant nodes
    yfine=
    # Interpolate on the Chebishev nodes
    ychebfine =
    # Compute the approximation of the error: use np.max and np.abs of yfine-f2(xfine)
    errmax[k]=
    # Repeat for the interpolation on Chebishev nodes
    errmaxCh[k]=

# Plot in a semi-logarithmic plot
plt.semilogy(np.linspace(1,nmax,nmax-1),errmax[0:nmax-1],'k')
plt.semilogy(np.linspace(1,nmax,nmax-1),errmaxCh[0:nmax-1],'r')
```

In [ ]:
```
#
# Interpolation of f3 on [0,pi/4]
#
# n=10
#
#Interval of approximation
a=0
b=np.pi/4


# Total number of experiments
nmax=20

# Making the fine grid
xfine=np.linspace(a, b, num=len(x)*10)
yfine=f3(xfine)


errmax=np.zeros(nmax)
errmaxCh=np.zeros(nmax)
for k in range(nmax):
    # Make the grid with equidistant nodes
    x=np.linspace(a, b, num=k+1)
    # Evaluate f3 on the grid
    y=f3(x)
    # Make the grid with Chebishev nodes on [-1,1]
    xcheb=np.zeros(k+1)
    for j in range(k+1):
        xcheb[j]=

    # Transform the Chebishev nodes to the isterval [a,b]
    abxcheb=
    # Evaluate f3 on the Chebisev nodes
    ycheb=f3(abxcheb)
    # Interpolate oo the equidistant nodes
    yfine=
    # Interpolate on the Chebishev nodes
    ychebfine=
    # Compute the approximation of the error
    errmax[k]=
    errmaxCh[k]=

# Plot in a semi-logarithmic plot
plt.semilogy(np.linspace(1,nmax,nmax-1),errmax[0:nmax-1],'k')
plt.semilogy(np.linspace(1,nmax,nmax-1),errmaxCh[0:nmax-1],'r')
```

Piecewise-polynomial interpolation

c) Suppose f is continuous and is the function you want to approximate, here we want to obtain a piece-wise polynomial and continuous approximation of f on $[a, b]$.

The interval $[a, b]$ is subdivided in the disjoint union of K subintervals $a = v_0 < v_1 < \cdots < v_K = b$. Then we have implemented a method that performs Lagrangian interpolation on $n + 1$ nodes on each subinterval $[v_i, v_{i+1}]$ (and we are using interpolate(, ,)).
We use equidistant nodes on the subintervals. Fix $n = 1, 2, \ldots, 10$.

The function "piecewiseinterpolation(x, a, b, K, n, f)" implements piecewise-polynomial interpolation on the interval [a,b] with K subintervals and polynomials of degree n on each subinterval. Assume the piecewise-polynomial approximation you obtain is called $\tilde{f}$.

Make a plot of the interpolation error $\max_{x \in [a,b]} |f(x) - \tilde{f}(x)|$ as a function of K, give numerical evidence that the method converges as $K \to \infty$.

In [ ]:
```
# Piecewise polynomial interpolation
# Test with the Runge function
#####################
def piecewiseinterpolation(x, a, b, K, n, f):
    v=np.linspace(a, b, num=K+1)
    yv=f(v)
    yfine=np.zeros(K*n*10+1)
    xfine=np.zeros(K*n*10+1)
    for k in range(K):
        xvk=np.linspace(v[k], v[k+1], num=n+1)
        yvk=f(xvk)
        xvkfine=np.linspace(v[k], v[k+1], num=(n*10+1))
        xfine[k*n*10:(k+1)*n*10+1]=xvkfine[0:n*10]
        yvkfine=interpolate(xvkfine,xvk,yvk)
        yfine[k*n*10:(k+1)*n*10]=yvkfine[0:n*10]
        if k==K-1:
            xfine[(k+1)*n*10]=b
            yfine[(k+1)*n*10]=yvkfine[n*10]


    return xfine, yfine

a=-5
b=5

Kmax=100
errmaxEQK=np.zeros(Kmax)
for K in range(Kmax):
    # Use the piecewiseinterpolation function  provided here. Try different values of the plynomial degree
    xf, yf =
    # Compute the error as in the previous exercise
    errmaxEQK[K]=

#Plot the results in a semi-logarithmic plot.
Nelements=np.linspace(1,Kmax,Kmax)
plt.semilogy(Nelements[0:Kmax-1], errmaxEQK[0:Kmax-1],'k')
```

d) In this part you find a suggestion for solving the exercise 2) in the multiple choice exerciseset 10. You can simply use the function interpolate(, ,) with appropriate input and requiring appropriate output and you can find which is the correct answer. Complete the code below to get your answer.

In [ ]:
```
#data from exercise 2) flervalgsoppgave 10
xx=np.array([1975, 1980, 1985, 1990])
fv=np.array([72.8, 74.2, 75.2, 76.4])
gv= np.array([70.2, 70.2, 70.3, 71.2])
#Output should be given in
xr= np.array([1977, 1983, 1988])

#r1 = interpolate(,,)
#r2 = interpolate(,,)

#print(r1)
#print(r2)
```

e) The following code is a simple implementation of Adaptive Simpson Quadrature. We want to make sure that it is correct.

- Explain what the code does.
- Make a test that shows that the code works as expected, use functions for which you know the exact integral and check that the code gives the correct answer.
- How can you garantee that the code indeed works?

In [ ]:
```
def adaptiveS(f,a,b,TOL):
    I0 = S(f,a,b)
    print(I0)
    c = (b+a)/2
    print(c)
    It = S(f,a,c) + S(f,c,b)
    if 1/15*abs(It-I0) <= TOL:
        I=It+1/15*(It-I0)
        It=I1
    else:
        It=adaptiveS(f,a,c,TOL/2)+adaptiveS(f,c,b,TOL/2)
    return It

def S(f,a,b):
    SS = (b-a)/6*(f(a)+4*f((a+b)/2)+f(b))
    return SS

def f0(x):
    return np.exp(1)**(3*x)*np.sin(2*x)

def f1(x):
    return x**10
```