Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Homework 10

Due Wednesday, April 12

In [1]:
```
using PyPlot    # Packages needed
```

Description

In this homework, you will make some extensions to the `MyPoly` type in the lecture notes.

Remember to write your functions in the style of generic programming, as discussed in the lecture notes. That is, they should correctly handle all types of coefficient vectors, including rational and complex numbers. This is almost automatic, but sometimes the function `eltype` is useful to ensure the right type.

First, we define the relevant functions below, with some simplifications (in particular we remove the `var` field and assume that the independent variable is x).

In [2]:
```
struct MyPoly
    c
end

function degree(p::MyPoly)
    ix1 = findfirst(p.c .!= 0)
    if ix1 == nothing
        return 0
    else
        return length(p.c) - ix1
    end
end

function Base.show(io::IO, p::MyPoly)
    d = degree(p)
    print(io, "MyPoly(")
    for k = d:-1:0
        coeff = p.c[end-k]
        if coeff == 0 && d > 0
            continue
```

```
            continue
        end
        if k < d
            if isa(coeff, Real)
                if coeff > 0
                    print(io, " + ")
                else
                    print(io, " - ")
                end
                coeff = abs(coeff)
            else
                print(io, " + ")
            end
        end
        if isa(coeff, Real)
            print(io, coeff)
        else
            print(io, "($coeff)")
        end
        if k == 0
            continue
        end
        print(io, "·x")
        if k > 1
            print(io, "^", k)
        end
    end
    print(io, ")")
end

function (p::MyPoly)(x)
    d = degree(p)
    v = p.c[end-d]
    for cc = p.c[end-d+1:end]
        v = v*x + cc
    end
    return v
end

function PyPlot.plot(p::MyPoly, xlim=[-2,2])
    xx = collect(range(xlim[1], xlim[2], length=100))
    plot(xx, p.(xx))
    xlabel("x")
end

function Base.:+(p1::MyPoly, p2::MyPoly)
    d1 = length(p1.c)
    d2 = length(p2.c)
    d = max(d1,d2)
    c = [fill(0, d-d1); p1.c] + [fill(0, d-d2); p2.c]
    return MyPoly(c)
```

```
end

function Base.:-(p1::MyPoly, p2::MyPoly)
    return p1 + MyPoly(-p2.c)
end

function Base.:*(a, p::MyPoly)
    newp = deepcopy(p)
    newp.c .*= a
    return newp
end

function Base.:*(p::MyPoly, a)
    return a*p
end
```

Problem 1(a)

Implement multiplication of two polynomials by overloading the `*` operator.

Test your function using the code below.

In [3]:

```
function Base.:*(p1::MyPoly, p2::MyPoly)
    deg = degree(p1) + degree(p2) + 1
    v = zeros(eltype(p1.c), deg)
    sizep2 = length(p2.c)
    for i = sizep2:-1:(sizep2 - degree(p2))
        v[end - degree(p1) + (i - sizep2):end + (i - sizep2)] += p
    end
    MyPoly(v)
end
```

In [4]:

```
p1 = MyPoly([1,-2])
p2 = MyPoly([4,3,1,1])
display(p1 * p2)  # Should have integer coefficients

p1 = MyPoly([2//3, -3//7])
p2 = MyPoly([-1//2, 1//3, -1//4])
display(p1 * p2)  # Should have rational coefficients
```

```
MyPoly(4⋅x^4 - 5⋅x^3 - 5⋅x^2 - 1⋅x - 2)

MyPoly(-1//3⋅x^3 + 55//126⋅x^2 - 13//42⋅x + 3//28)
```

Problem 1(b)

Implement a new constructor for the `MyPoly` type, which creates a polynomial from a given vector of roots (and with leading term 1). That is, for a vector r with d roots, we define the degree d polynomial

$$p(x) = \prod_{k=1}^{d} (x - r_k)$$

To make sure we can still use the old syntax of initializing by the coefficients `c`, overload `MyPoly` with a parameter named `roots`:

```
function MyPoly(; roots)
    # Implement function here
end
```

Test your function using the code below.

In [5]:
```
function MyPoly(; roots)
    polys = []
    for r in roots
        push!(polys, MyPoly([1, -r]))
    end
    return prod(polys)
end
```

Out[5]: MyPoly

In [6]:
```
p = MyPoly(roots=[-3,-2,0,1,1,4])
display(p)  # Should have integer coefficients

p = MyPoly(roots=[-7//3,-2//1,0,1//2,1//2,3//2])
display(p)  # Should have rational coefficients
```

```
MyPoly(1⋅x^6 - 1⋅x^5 - 15⋅x^4 + 5⋅x^3 + 34⋅x^2 - 24⋅x)

MyPoly(1//1⋅x^6 + 11//6⋅x^5 - 53//12⋅x^4 - 107//24⋅x^3 + 157//24⋅x^2
- 7//4⋅x)
```

Problem 1(c)

Implement a function `differentiate` which returns the derivative of a polynomial.

Test your function using the code below.

In [7]:
```
function differentiate(p::MyPoly)
    l = length(p.c)
    for i = 1:l - 1
        p.c[end - i] *= i
    end
    v = p.c[1:end - 1]
    return MyPoly(v)
end
```

Out[7]:
```
differentiate (generic function with 1 method)
```

In [8]:
```
differentiate(p)
```

Out[8]:
```
MyPoly(6//1·x^5 + 55//6·x^4 - 53//3·x^3 - 107//8·x^2 + 157//12·x - 7//4)
```

Problem 1(d)

Implement a function `integrate` which returns the (indefinite) integral of a polynomial, with the constant term = 0.

Test your function using the code below.

In [9]:
```
function integrate(p::MyPoly)
    l = length(p.c)
    for i = 1:l - 1
        p.c[end - i] /= i
    end
    return MyPoly(p.c)
end
```

Out[9]:
```
integrate (generic function with 1 method)
```

In [10]:
```
integrate(p)
```

Out[10]:
```
MyPoly(1//1·x^6 + 11//6·x^5 - 53//12·x^4 - 107//24·x^3 + 157//24·x^2 - 7//4·x)
```

Problem 2

In this problem you will use the polynomial type to compute Lagrange polynomials for a set of nodes, and compute so-called elemental matrices that appear in the finite element discretization of PDEs.

Problem 2(a)

Implement a function `LagrangePolynomials(s)` where `s` is a vector of n numbers, which returns a vector of n polynomials $L_k(x)$, $k = 1, \ldots, n$, of degree $d = n - 1$ such that

$$L_k(s_j) = \delta_{kj} = \begin{cases} 1 & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases}$$

Hint: Note that a polynomial $L_k(x)$ is zero at the $n - 1$ points s_j, $j \neq k$. Use the `roots` constructor of `MyPoly` to create a polynomial with these roots, then evaluate it and scale to make $L_k(s_k) = 1$.

Test your function using the code below.

In [11]:

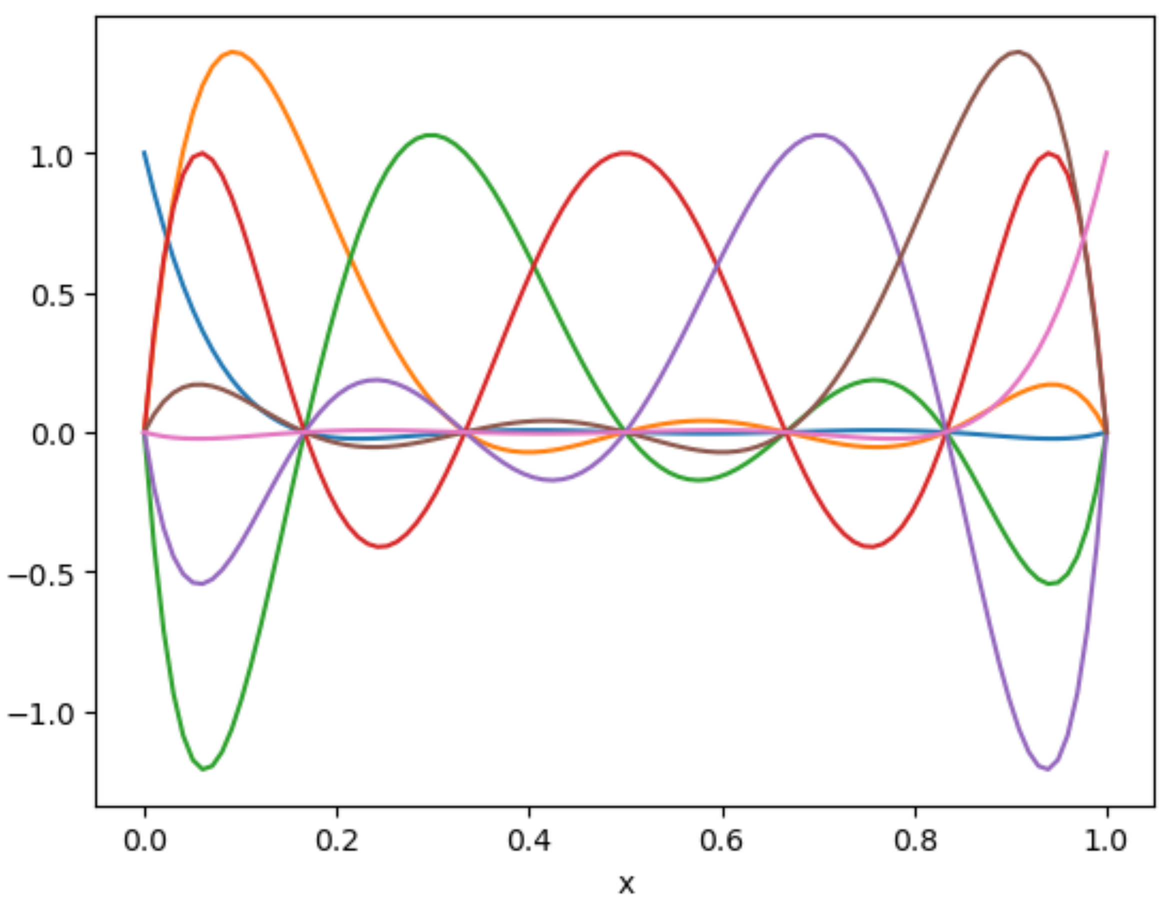
```
function LagrangePolynomials(s)
    l = length(s)
    polys = []
    s2 = [0;s;0]

    for i = 1:l
        s_new = append!(s2[1:i], s2[i + 2:end])
        p1 = MyPoly(roots = s_new[2:end - 1])
        c = p1(s[i])
        p2 = MyPoly(p1.c ./ c)
        push!(polys, p2)
    end
    display(polys)

    return polys
end
```

Out[11]:

```
LagrangePolynomials (generic function with 1 method)
```

In [12]:
```
Ls = LagrangePolynomials((0:6) / 6)
plot.(Ls, Ref([0,1]));
```

```
7-element Vector{Any}:
 MyPoly(64.8·x^6 - 226.8·x^5 + 315.0·x^4 - 220.5·x^3 + 81.2·x^2 - 14.
700000000000001·x + 1.0)
 MyPoly(-388.79999999999933·x^6 + 1295.999999999998·x^5 - 1673.999999
999997·x^4 + 1043.9999999999982·x^3 - 313.1999999999994·x^2 + 35.9999
99999999936·x)
 MyPoly(971.9999999999958·x^6 - 3077.999999999987·x^5 + 3698.99999999
9984·x^4 - 2074.499999999991·x^3 + 526.4999999999977·x^2 - 44.9999999
999998·x)
 MyPoly(-1295.999999999971·x^6 + 3887.9999999999127·x^5 - 4355.999999
999902·x^4 + 2231.99999999995·x^3 - 507.9999999999886·x^2 + 39.999999
9999991·x)
 MyPoly(972.0000000000241·x^6 - 2754.000000000068·x^5 + 2889.00000000
0072·x^4 - 1381.5000000000343·x^3 + 297.0000000000074·x^2 - 22.500000
000000558·x)
 MyPoly(-388.8000000000087·x^6 + 1036.8000000000231·x^5 - 1026.000000
000023·x^4 + 468.0000000000105·x^3 - 97.20000000000218·x^2 + 7.200000
000000161·x)
 MyPoly(64.8·x^6 - 162.0·x^5 + 153.0·x^4 - 67.50000000000001·x^3 + 13
.700000000000003·x^2 - 1.0·x)
```



Problem 2(b)

In the finite element method (FEM), so-called mass matrices M and stiffness matrices K are defined as follows:

$$M_{ij} = \int_0^1 L_i(x)L_j(x)\,dx, \qquad i,j = 1,\ldots,n$$
$$K_{ij} = \int_0^1 \frac{dL_i}{dx}\frac{dL_j}{dx}\,dx, \qquad i,j = 1,\ldots,n$$

Write two functions `mkM(Ls)` and `mkK(Ls)` which computes and returns these matrices, for a given vector `Ls` of Lagrange polynomials.

Test your function using the code below. Note that in this case, the `s` vector is *rational* and all your outputs should also be rational.

In [13]:
```
function mkM(Ls)
    l = length(Ls)
    M = zeros(l, l)
    for i = 1:l
        for j = 1:l
            poly_prod = Ls[i] * Ls[j]
            poly_int = integrate(poly_prod)
            M[i,j] = poly_int(1) - poly_int(0)
        end
    end
    return M
end
function mkK(Ls)
    l = length(Ls)
    M = zeros(l, l)
    for i = 1:l
        for j = 1:l
            poly_der1 = differentiate(Ls[i])
            poly_der2 = differentiate(Ls[j])
            poly_prod = poly_der1 * poly_der2
            M[i,j] = poly_prod(1) - poly_prod(0)
        end
    end
    return M
end
```

Out[13]:
```
mkK (generic function with 1 method)
```

In [14]:

```
d = 4
Ls = LagrangePolynomials((0:d) // d)
M = mkM(Ls)
K = mkK(Ls)
display(Ls)
display(M)
display(K)
```

```
5-element Vector{Any}:
 MyPoly(32//3·x^4 - 80//3·x^3 + 70//3·x^2 - 25//3·x + 1//1)
 MyPoly(-128//3·x^4 + 96//1·x^3 - 208//3·x^2 + 16//1·x)
 MyPoly(64//1·x^4 - 128//1·x^3 + 76//1·x^2 - 12//1·x)
 MyPoly(-128//3·x^4 + 224//3·x^3 - 112//3·x^2 + 16//3·x)
 MyPoly(32//3·x^4 - 16//1·x^3 + 22//3·x^2 - 1//1·x)

5-element Vector{Any}:
 MyPoly(33554432//3·x^4 - 1574640//1·x^3 + 71680//3·x^2 - 25//3·x + 1
//1)
 MyPoly(-134217728//3·x^4 + 5668704//1·x^3 - 212992//3·x^2 + 16//1·x)
 MyPoly(67108864//1·x^4 - 7558272//1·x^3 + 77824//1·x^2 - 12//1·x)
 MyPoly(-134217728//3·x^4 + 4408992//1·x^3 - 114688//3·x^2 + 16//3·x)
 MyPoly(33554432//3·x^4 - 944784//1·x^3 + 22528//3·x^2 - 1//1·x)

5×5 Matrix{Float64}:
 -3.47354     1.0328    -0.679365   0.287831  -0.0566138
  1.0328      2.09947   -0.761905   0.338624  -0.042328
 -0.679365   -0.761905   1.02857   -0.253968   0.0
  0.287831    0.338624  -0.253968   0.474074   0.042328
 -0.0566138  -0.042328   0.0        0.042328   0.0566138

5×5 Matrix{Float64}:
 -53.7778      -618.667       11128.0         -82851.6          2.144
56e5
  -9.55355e6      2.87164e6      -3.70052e6        1.68895e7   -2.738
76e7
   6.55149e8     -1.57691e10      6.4006e9        -7.84583e8    1.040
3e9
  -1.24818e10     2.49466e11     -1.75441e12       1.25179e12  -1.509
78e10
   7.00235e10    -1.2719e12       8.39021e12      -2.40576e13   2.546
54e13
```