```
> restart;
  with(plots):
  with(LinearAlgebra):
  with(IntegrationTools):
```

Finite element methods in one spatial dimension

The purpose of this worksheet is to develop a finite element algoirthm to solve partial differential equations of the form

$$\rho(x)\frac{\partial^2 u}{\partial t^2} + \lambda(x)\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[P(x)\frac{\partial u}{\partial x}\right] + Q(x)u - R(x), \quad u(t,-1) = \alpha, \quad u(t,1) = \beta.$$

In these equations, $\{\rho, \lambda, P, Q, R\}$ are known functions of x while $\{\alpha, \beta\}$ are known constants.

Weak form of the problem

Here is the PDE we want to solve:

```
> pde := rho(x)*Diff(u(t,x),t,t) + lambda(x)*Diff(u(t,x),t) - Diff(P(x)*Diff(u(t,x),x),x) -
  Q(x)*u(t,x) + R(x);
```

$$pde := \rho(x)\left(\frac{\partial^2}{\partial t^2}u(t,x)\right) + \lambda(x)\left(\frac{\partial}{\partial t}u(t,x)\right) - \left(\frac{\partial}{\partial x}\left(P(x)\left(\frac{\partial}{\partial x}u(t,x)\right)\right)\right) - Q(x)\,u(t,x) + R(x) \qquad \textbf{(1.1)}$$

We do not want Maple to expand out the derivatives in the third term, so we use the inert differentiation command **Diff**. The first step in the finite element solution of this PDE is deriving the so-called "weak form" of the equation. This involves multiplying the PDE by a test function of x, which we denote by $\phi_i(x)$, and integrating over $[-1, 1]$. Note that right now $\phi_i(x)$ is an arbitrary function, but in the next section we will identify it with one member of an N-dimensional "basis": $\{\phi_i(x)\}_{i=1}^{N}$.

```
> eq1 := int(pde*phi[i](x),x=-1..1);
  eq2 := Expand(eq1);
```

$$eq1 := \int_{-1}^{1}\left(\rho(x)\left(\frac{\partial^2}{\partial t^2}u(t,x)\right) + \lambda(x)\left(\frac{\partial}{\partial t}u(t,x)\right) - \left(\frac{\partial}{\partial x}\left(P(x)\left(\frac{\partial}{\partial x}u(t,x)\right)\right)\right) - Q(x)\,u(t,x) + R(x)\right)\phi_i(x)\,dx$$

$$eq2 := \int_{-1}^{1}\phi_i(x)\,\rho(x)\left(\frac{\partial^2}{\partial t^2}u(t,x)\right)dx + \int_{-1}^{1}\phi_i(x)\,\lambda(x)\left(\frac{\partial}{\partial t}u(t,x)\right)dx - \left(\int_{-1}^{1}\phi_i(x)\left(\frac{\partial}{\partial x}\left(P(x)\left(\frac{\partial}{\partial x}u(t,x)\right)\right)\right)\right) \qquad \textbf{(1.2)}$$

$$\left. dx \right) - \left(\int_{-1}^{1} \phi_i(x)\, Q(x)\, u(t,x)\, dx \right) + \int_{-1}^{1} \phi_i(x)\, R(x)\, dx$$

The third term in **(1.2)** can be integrated by parts:

```
> eq3 := applyop(u->Parts(u,phi[i](x)),3,eq2);
```

$$eq3 := \int_{-1}^{1} \phi_i(x)\, \rho(x) \left(\frac{\partial^2}{\partial t^2} u(t,x) \right) dx + \int_{-1}^{1} \phi_i(x)\, \lambda(x) \left(\frac{\partial}{\partial t} u(t,x) \right) dx - P(1)\, \mathrm{D}_2(u)(t,1)\, \phi_i(1) + P(-1)\, \mathrm{D}_2(u)(t, -1)\, \phi_i(-1) + \int_{-1}^{1} P(x) \left(\frac{\partial}{\partial x} u(t,x) \right) \left(\frac{d}{dx} \phi_i(x) \right) dx - \left(\int_{-1}^{1} \phi_i(x)\, Q(x)\, u(t,x)\, dx \right) + \int_{-1}^{1} \phi_i(x)\, R(x)\, dx \qquad \textbf{(1.3}$$

Now, let us assume that our test function $\phi_i(x)$ vanishes at either end of the region of integration:

```
> eq4 := [phi[i](-1)=0,phi[i](1)=0];
  eq5 := Expand(convert(subs(eq4,eq3),diff));
```

$$eq4 := [\phi_i(-1) = 0, \phi_i(1) = 0]$$

$$eq5 := \int_{-1}^{1} \phi_i(x)\, \rho(x) \left(\frac{\partial^2}{\partial t^2} u(t,x) \right) dx + \int_{-1}^{1} \phi_i(x)\, \lambda(x) \left(\frac{\partial}{\partial t} u(t,x) \right) dx + \int_{-1}^{1} P(x) \left(\frac{\partial}{\partial x} u(t,x) \right) \left(\frac{d}{dx} \phi_i(x) \right) dx - \left(\int_{-1}^{1} \phi_i(x)\, Q(x)\, u(t,x)\, dx \right) + \int_{-1}^{1} \phi_i(x)\, R(x)\, dx \qquad \textbf{(1.4}$$

This is the "weak form" of the original PDE **(1.1)**, and it is the central equation of our finite element analysis. Our goal will to be to find a solutions of **(1.4)** for a particular choice of N basis functions $\{\phi_i(x)\}_{i=1}^{N}$; i.e., **(1.4)** represents one of a system of N equations. In what sense is equation **(1.4)** weak? For a given choice of basis functions, there will be a solution $u(t, x)$ that satisfies the associated system of equations, yet are not solutions of the PDE **(1.1)**. However, for a reasonable choice of basis, the hope is that the solutions of **(1.4)** are good approximations to the solutions of **(1.1)**. This is the essence of the finite element analysis, we seek exact solutions of the weak equations that approximate the solutions of the "strong" equations **(1.1)**.

Basis functions

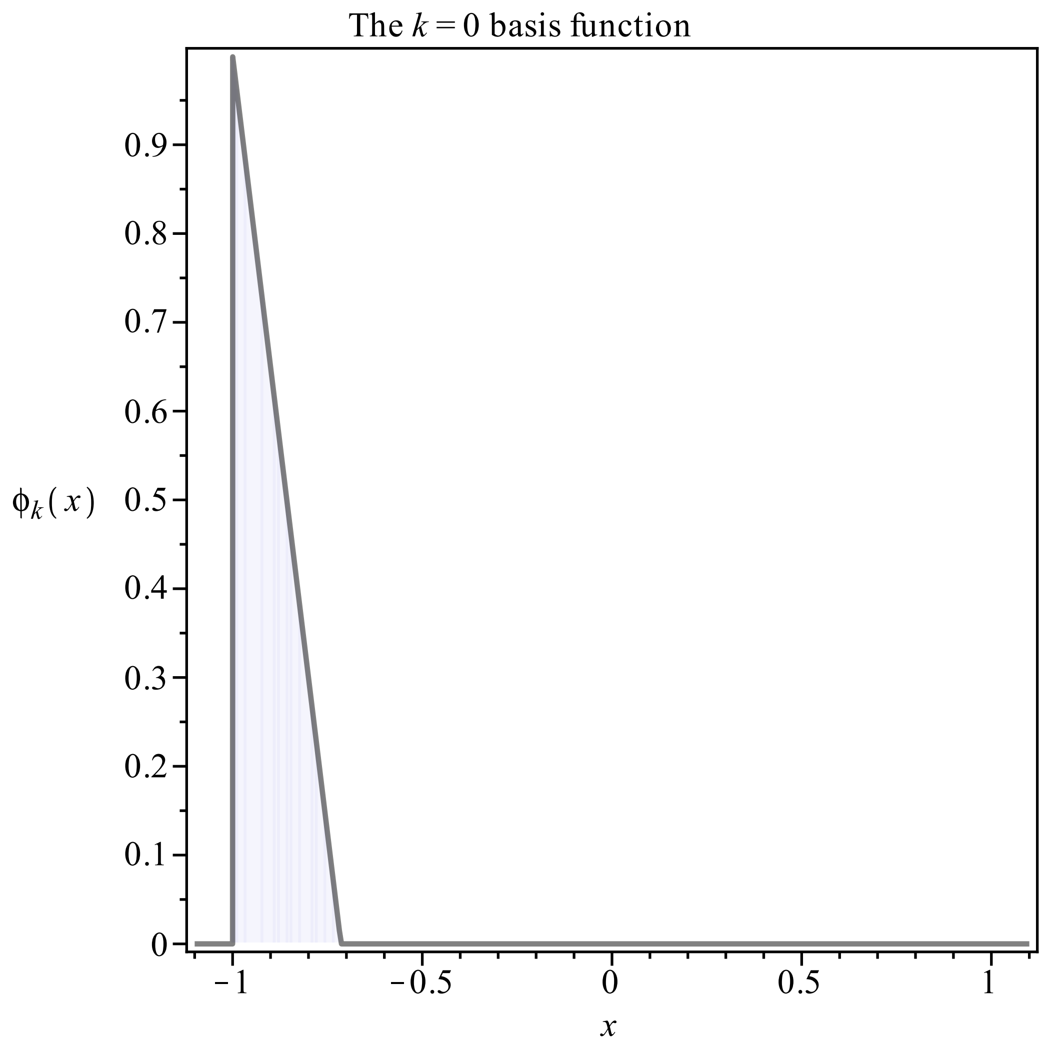
We cannot proceed any further until we choose a basis. We will restrict ourselves to simplest choice: piecewise linear functions. (Of course, many other choices of basis are possible.) To be more specific, we introduce a lattice on the interval $[-1, 1]$ of the form

$x_i = -1 + \frac{2\,i}{N+1}$, with $i = 0 \ldots N$. With this choice, $x_0 = -1$ and $x_{N+1} = 1$. A basis function will be a piecewise linear function defined on $[-1, 1]$ whose value at each position on the lattice is

$$\phi_i(x_j) = \begin{cases} 0 & x_j = x_i \\ 1 & x_j \neq x_i \end{cases}$$

Here is some code that defines the basis for a particular choice of N and plots the functions in a movie. Notice the use of **piecewise** to define each member of the basis.

```
> phi_def := 'phi_def':
  N := 6:
  X := i -> -1 + 2*i/(N+1):
  phi_def[0] := phi[0](x) = piecewise(x>X(0) and x < X(1),(X(1)-x)/(X(1)-X(0))):
  for i from 1 to N do:
     phi_def[i] := phi[i](x) = piecewise(x>X(i-1) and x <= X(i), (x-X(i-1))/(X(i)-X(i-1)),
  x>X(i) and x < X(i+1),(X(i+1)-x)/(X(i+1)-X(i)));
  od:
  phi_def[N+1] := phi[N+1](x) = piecewise(x>X(N) and x < X(N+1),(x-X(N))/(X(N+1)-X(N))):
  phi_def := convert(phi_def,list):
  i := 'i':
  display([seq(plot([rhs(phi_def[i])$2],x=-1.1..1.1,color=[black,"Lavender"],thickness=[2,
  0],transparency=0.5,filled=[false,true],axes=boxed,labels=[x,typeset(phi[k](x))],title=
  typeset("The ",k=i-1," basis function")),i=1..N+2)],insequence=true);
```



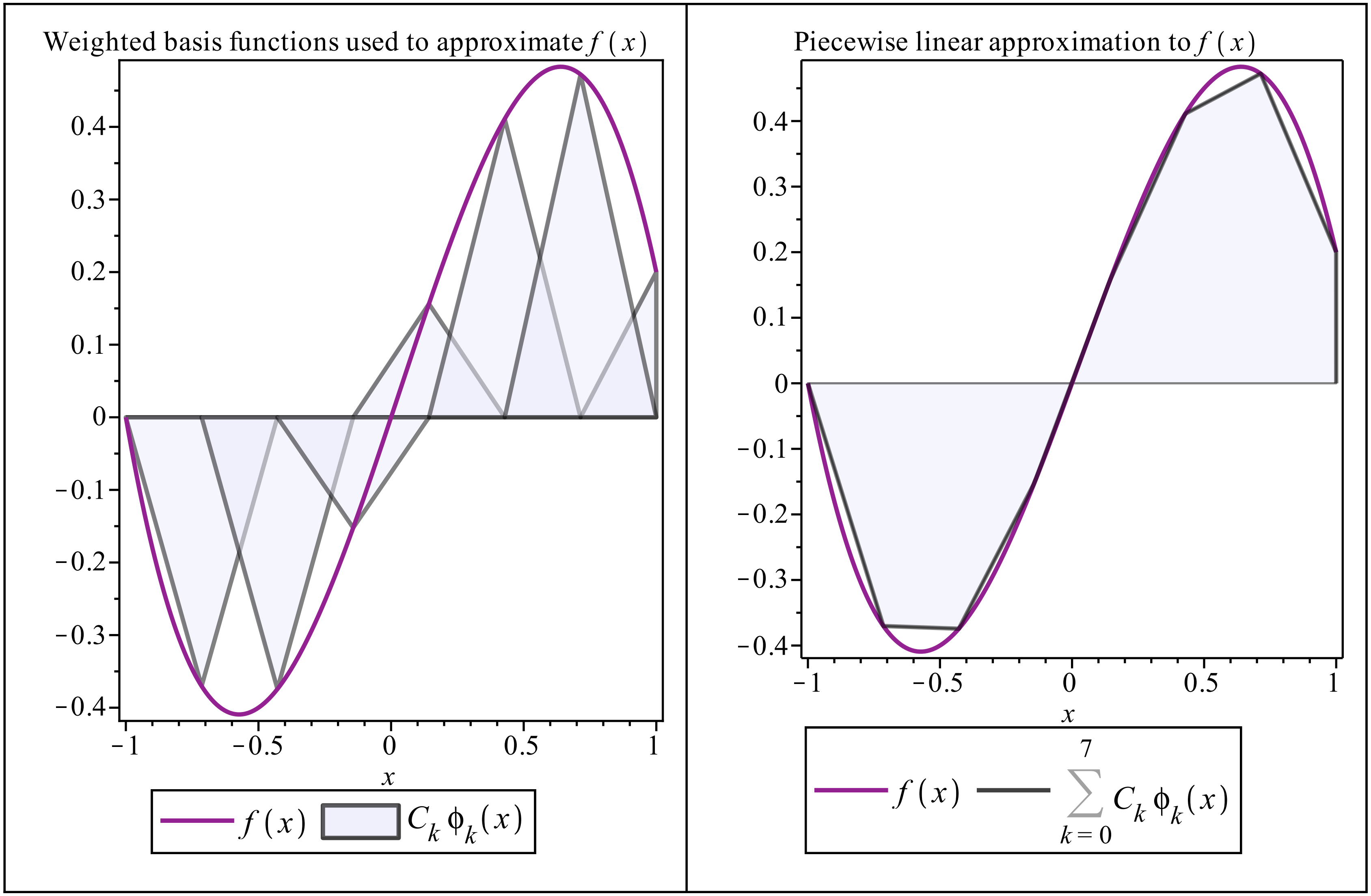
Theses basis functions are the "finite elements" we will use to construct the numeric solution to **(1.1)**. Note that the basis can be used to obtain a piecewise linear approximation to an function $f(x)$ defined on $[-1, 1]$:

$$f(x) \approx \sum_{i=0}^{N} C_i \phi_i(x), \quad C_i = f\left(x_i\right).$$

The LHS will equal the RHS when at the lattice points (i.e., when $x = x_i$). In between lattice points, the RHS will be a continuous linear function. Here are some plots showing how the decomposition works:

```
> f := x -> x*(11/10-x)*(1+x);
  c := Array(0..N+1,[seq(f(X(i)),i=0..N+2)]):
  Data1 := [[[X(0),0],[X(1),0],[X(0),c[0]]],seq([[X(i-1),0],[X(i+1),0],[X(i),c[i]]],i=1..N),
  [[X(N),0],[X(N+1),0],[X(N+1),c[N+1]]]]:
  Data2 := subs(phi_def,add(c[i]*phi[i](x),i=0..N+1)):
  Data3 := [[-1,0],seq([X(i),c[i]],i=0..N+1),[1,0]]:
  p1 := plot(f(x),x=-1..1,legend=[''''f(x)''''],thickness=2,color="Purple"):
  p2 := polygonplot(Data1,color="Lavender",axes=boxed,transparency=0.5,thickness=2,legend=
  typeset(C[k]*phi[k](x))):
  p3 := plot([Data2],x=-1..1,color=black,axes=boxed,legend=[typeset(Sum(C[k]*phi[k](x),k=0..
  N+1))],transparency=0.5,thickness=2):
  p4 := polygonplot(Data3,color="Lavender",transparency=0.5):
  display(Array([display([p1,p2],title=typeset("Weighted basis functions used to approximate
  ",''f(x)'')),display([p1,p3,p4],title=typeset("Piecewise linear approximation to ",''f(x)
  ''))]));
```

$$f := x \rightarrow x \left(\frac{11}{10} - x\right) (x + 1)$$



Now, let's us write down a piewise linear approximation to the solution of the PDE **(1.1)** using these basis functions:

```
> N := 'N':
  eq6 := u(t,x) = alpha*phi[0](x) + Sum(a[j](t)*phi[j](x),j=1..N) + beta*phi[N+1](x);
```

$$eq6 := u(t,x) = \alpha\,\phi_0(x) + \sum_{j=1}^{N} a_j(t)\,\phi_j(x) + \beta\,\phi_{N+1}(x) \qquad \textbf{(2.1}$$

Notice that we have enforced the boundary conditions $u(t,-1)=\alpha$ and $u(t,+1)=\beta$ by forcing the coefficients of $\phi_0(x)$ and $\phi_{N+1}(x)$ to be α and β, respectively. Also note that all the time dependence of the solution is carried by the other coefficients $a_j(t)$. Our goal will to be to solve for these coefficients.

Matrix form

We now substitute the decomposition of $u(t,x)$ in terms of basis functions **(2.1)** into the weak form of the PDE **(1.4)**:

```
> eq7 := Expand(convert(subs(eq6,eq5),diff));
```

$$eq7 := \int_{-1}^{1} \phi_i(x)\,\rho(x)\left(\sum_{j=1}^{N}\left(\frac{d^2}{dt^2}a_j(t)\right)\phi_j(x)\right)dx + \int_{-1}^{1}\phi_i(x)\,\lambda(x)\left(\sum_{j=1}^{N}\left(\frac{d}{dt}a_j(t)\right)\phi_j(x)\right)dx + \alpha\left(\int_{-1}^{1}P(x)\left(\frac{d}{dx}\phi_i(x)\right)\left(\frac{d}{dx}\phi_0(x)\right)dx\right) + \int_{-1}^{1}P(x)\left(\frac{d}{dx}\phi_i(x)\right)\left(\sum_{j=1}^{N}a_j(t)\left(\frac{d}{dx}\phi_j(x)\right)\right)dx + \beta\left(\int_{-1}^{1}P(x)\left(\frac{d}{dx}\phi_i(x)\right)\left(\frac{d}{dx}\phi_{N+1}(x)\right)dx\right) - \alpha\left(\int_{-1}^{1}\phi_i(x)\,Q(x)\,\phi_0(x)\,dx\right) - \left(\int_{-1}^{1}\phi_i(x)\,Q(x)\left(\sum_{j=1}^{N}a_j(t)\,\phi_j(x)\right)dx\right) - \beta\left(\int_{-1}^{1}\phi_i(x)\,Q(x)\,\phi_{N+1}(x)\,dx\right) + \int_{-1}^{1}\phi_i(x)\,R(x)\,dx \tag{3.1}$$

We now evaulate this expression for each of the $\phi_i(x)$ basis functions with an unknown coefficient; i.e., for $i = 1 \ldots N$. It is not difficult to see that this yields a *matrix* ODE for the unknown coefficients:

$$M\frac{d^2\mathbf{a}}{dt^2} + C\frac{d\mathbf{a}}{dt} + K\mathbf{a} = \mathbf{f}, \qquad \mathbf{a} = \begin{bmatrix} a_1(t) \\ \vdots \\ a_N(t) \end{bmatrix}$$

$$M = \begin{bmatrix} m_{1,1} & \cdots & m_{1,N} \\ \vdots & & \vdots \\ m_{N,1} & \cdots & m_{N,N} \end{bmatrix}, \quad C = \begin{bmatrix} c_{1,1} & \cdots & c_{1,N} \\ \vdots & & \vdots \\ c_{N,1} & \cdots & c_{N,N} \end{bmatrix}, \quad K = \begin{bmatrix} k_{1,1} & \cdots & k_{1,N} \\ \vdots & & \vdots \\ k_{N,1} & \cdots & k_{N,N} \end{bmatrix}, \quad \boldsymbol{f} = \begin{bmatrix} F_1 \\ \vdots \\ F_N \end{bmatrix},$$

where:

$$m_{i,j} = \int_{-1}^{1} \phi_i(x)\rho(x)\phi_j(x)\,\mathrm{d}x, \quad c_{i,j} = \int_{-1}^{1} \phi_i(x)\lambda(x)\phi_j(x)\,\mathrm{d}x, \quad k_{i,j} = \int_{-1}^{1} \left\{ \left[\frac{\mathrm{d}}{\mathrm{d}x}\phi_i(x) \right] P(x) \left[\frac{\mathrm{d}}{\mathrm{d}x}\phi_j(x) \right] - \phi_i(x) Q(x) \phi_j(x) \right\} \mathrm{d}x,$$

$$F_i = -\alpha\, k_{i,0} - \beta\, k_{i,N+1} - \int_{-1}^{1} \phi_i(x) R(x)\,\mathrm{d}x$$

Finite element methods were first used extensively in structural engineering, so the names of the above objects are inherited from that field: M is the mass matrix, C is the damping matrix, K is the stiffness matrix, and **f** is the applied force. The three matricies are symmetric, and since the basis functions $\phi_i(x)$ are only non-zero in the interval $[x_{i-1}, x_{i+1}]$ they will also be tridiagonal. Here are some procedures we can use to generate the elements of these objects:

```
> # this procedure generates an array of the basis function written as mappings (its output is used in the following):
GenerateBasis := proc(N)
    local X, phi, i:
    X := i -> -1 + 2*i/(N+1):
    phi := Array(0..N+1):
    phi[0] := unapply(piecewise(x>X(0) and x < X(1),(X(1)-x)/(X(1)-X(0))),x):
    for i from 1 to N do:
       phi[i] := unapply(piecewise(x>X(i-1) and x <= X(i), (x-X(i-1))/(X(i)-X(i-1)), x>X(i) and x < X(i+1),(X(i+1)-x)/(X(i+1)-X(i))),x);
    od:
    phi[N+1] := unapply(piecewise(x>X(N) and x < X(N+1),(x-X(N))/(X(N+1)-X(N))),x):
    phi:
end proc:


# this procedure generates elements of the mass matrix given the basis functions phi
m := proc(phi,rho,i,j):
    int(phi[i](x)*rho(x)*phi[j](x),x=-1..1):
end proc:
```

this procedure generates elements of the damping matrix given the basis functions **phi**

```
c := proc(phi,lambda,i,j):
    int(phi[i](x)*lambda(x)*phi[j](x),x=-1..1):
end proc:
```

this procedure generates elements of the stiffness matrix given the basis functions **phi**

```
k := proc(phi,P,Q,i,j):
    int(D(phi[i])(x)*P(x)*D(phi[j])(x)-phi[i](x)*Q(x)*phi[j](x),x=-1..1):
end proc:
```

this procedure generates components of the applied force given the basis functions **phi**

```
F := proc(phi,P,Q,R,i,alpha,beta):
    -alpha*k(phi,P,Q,i,0)-beta*k(phi,P,Q,i,rhs(ArrayDims(phi)))-int(phi[i](x)*R(x),x=-1.
.1):
end proc:
```

Here is an example of how the matrices and applied force look for a particular choice of parameters:

```
> Phi := GenerateBasis(4):
N := rhs(ArrayDims(Phi))-1;
l := 1;
rho := x -> 1;
lambda := x -> x;
P := x -> (1-x^2);
Q := x -> l*(l+1);
R := x -> 4;
alpha := (-1)^l;
beta := 1;
M := Matrix([seq([seq(m(Phi,rho,i,j),i=1..N)],j=1..N)]):
C := Matrix([seq([seq(c(Phi,lambda,i,j),i=1..N)],j=1..N)]):
K := Matrix([seq([seq(k(Phi,P,Q,i,j),i=1..N)],j=1..N)]):
f := Vector([seq(F(Phi,P,Q,R,i,alpha,beta,N),i=1..N)]):
pde;
'M'=M,'C'=C,'K'=K,'f'=f;
```

$$N := 4$$

$$l := 1$$

$$\rho := x \rightarrow 1$$

$$\lambda := x \rightarrow x$$

$$P := x \to 1 - x^2$$
$$Q := x \to l\,(l+1)$$
$$R := x \to 4$$
$$\alpha := -1$$
$$\beta := 1$$
$$\frac{\partial^2}{\partial t^2} u(t,x) + x\left(\frac{\partial}{\partial t} u(t,x)\right) - \left(\frac{\partial}{\partial x}\left((1-x^2)\left(\frac{\partial}{\partial x} u(t,x)\right)\right)\right) - 2\,u(t,x) + 4$$

$$M = \begin{bmatrix} \frac{4}{15} & \frac{1}{15} & 0 & 0 \\ \frac{1}{15} & \frac{4}{15} & \frac{1}{15} & 0 \\ 0 & \frac{1}{15} & \frac{4}{15} & \frac{1}{15} \\ 0 & 0 & \frac{1}{15} & \frac{4}{15} \end{bmatrix}, C = \begin{bmatrix} -\frac{4}{25} & -\frac{2}{75} & 0 & 0 \\ -\frac{2}{75} & -\frac{4}{75} & 0 & 0 \\ 0 & 0 & \frac{4}{75} & \frac{2}{75} \\ 0 & 0 & \frac{2}{75} & \frac{4}{25} \end{bmatrix}, K = \begin{bmatrix} \frac{12}{5} & -\frac{11}{5} & 0 & 0 \\ -\frac{11}{5} & 4 & -\frac{13}{5} & 0 \\ 0 & -\frac{13}{5} & 4 & -\frac{11}{5} \\ 0 & 0 & -\frac{11}{5} & \frac{12}{5} \end{bmatrix}, f = \begin{bmatrix} -\frac{13}{5} \\ -\frac{8}{5} \\ -\frac{8}{5} \\ -\frac{3}{5} \end{bmatrix} \quad (3.2$$

As claimed above, the matrices are symmetric and tridiagonal, which implies that the above method is not the most efficient way of calculating them. The following procedures are faster:

```
> mass := proc(phi,rho)
     local N, M, i:
     N := rhs(ArrayDims(phi))-1:
     M := Matrix(N,N,shape=symmetric,datatype=float):
     for i from 1 to N do:
        M[i,i] := evalf(m(phi,rho,i,i)):
        if (i<>N) then M[i,i+1] := evalf(m(phi,rho,i,i+1)) fi:
     od:
     M;
  end proc:

  damping := proc(phi,lambda)
     local N, C, i:
     N := rhs(ArrayDims(phi))-1:
     C := Matrix(N,N,shape=symmetric,datatype=float):
```

```
    for i from 1 to N do:
       C[i,i] := evalf(c(phi,lambda,i,i)):
       if (i<>N) then C[i,i+1] := evalf(c(phi,lambda,i,i+1)) fi:
    od:
    C;
end proc:

stiffness := proc(phi,P,Q)
    local N, K, i:
    N := rhs(ArrayDims(phi))-1:
    K := Matrix(N,N,shape=symmetric,datatype=float):
    for i from 1 to N do:
       K[i,i] := evalf(k(phi,P,Q,i,i)):
       if (i<>N) then K[i,i+1] := evalf(k(phi,P,Q,i,i+1)) fi:
    od:
    K;
end proc:

force := proc(phi,P,Q,R,alpha,beta)
    local N, K, i:
    N := rhs(ArrayDims(phi))-1:
    Vector([seq(evalf(F(phi,P,Q,R,i,alpha,beta,N)),i=1..N)]):
end proc:
```

Elliptic boundary value problems

In this section, we will concentrate on the time-independent case when $\rho(x) = \lambda(x) = 0$ in **(1.1)**; i.e., the equation we want to solve is:

$$0 = \frac{\mathrm{d}}{\mathrm{d}x}\left[P(x)\frac{\mathrm{d}u}{\mathrm{d}x}\right] + Q(x)u - R(x), \quad u(-1) = \alpha, \quad u(1) = \beta.$$

(We have re-writted $u(t,x) \mapsto u(x)$.) This is an ODE boundary value problem. In this case, the matrix equation we need to solve for the amplitudes $\mathbf{a}$ of the basis functions is

$$K\mathbf{a} = \mathbf{f}.$$

The following procedure calculates the coefficients and plots the finite element solution:

```
> EllipticSolver := proc(N,P,Q,R,alpha,beta)
     local Phi,K,i,f,ans:
```

```
    Phi := GenerateBasis(N):
    K := stiffness(Phi,P,Q):
    f := force(Phi,P,Q,R,alpha,beta):
    ans := LinearSolve(K,f);
    ans := Array(0..N+1,[alpha,seq(ans[i],i=1..N),beta]):
    plot(add(ans[i]*Phi[i](x),i=0..N+1),x=-1..1,color="Lavender",filled=true,legend="Finite
element solution",axes=boxed):
end proc:
```

We illustrate the output of the above code for the following example with exact solution **u_sol**. (This is actually Legendre's differential equation whose solutions are the Legendre functions $P_l(x)$.)

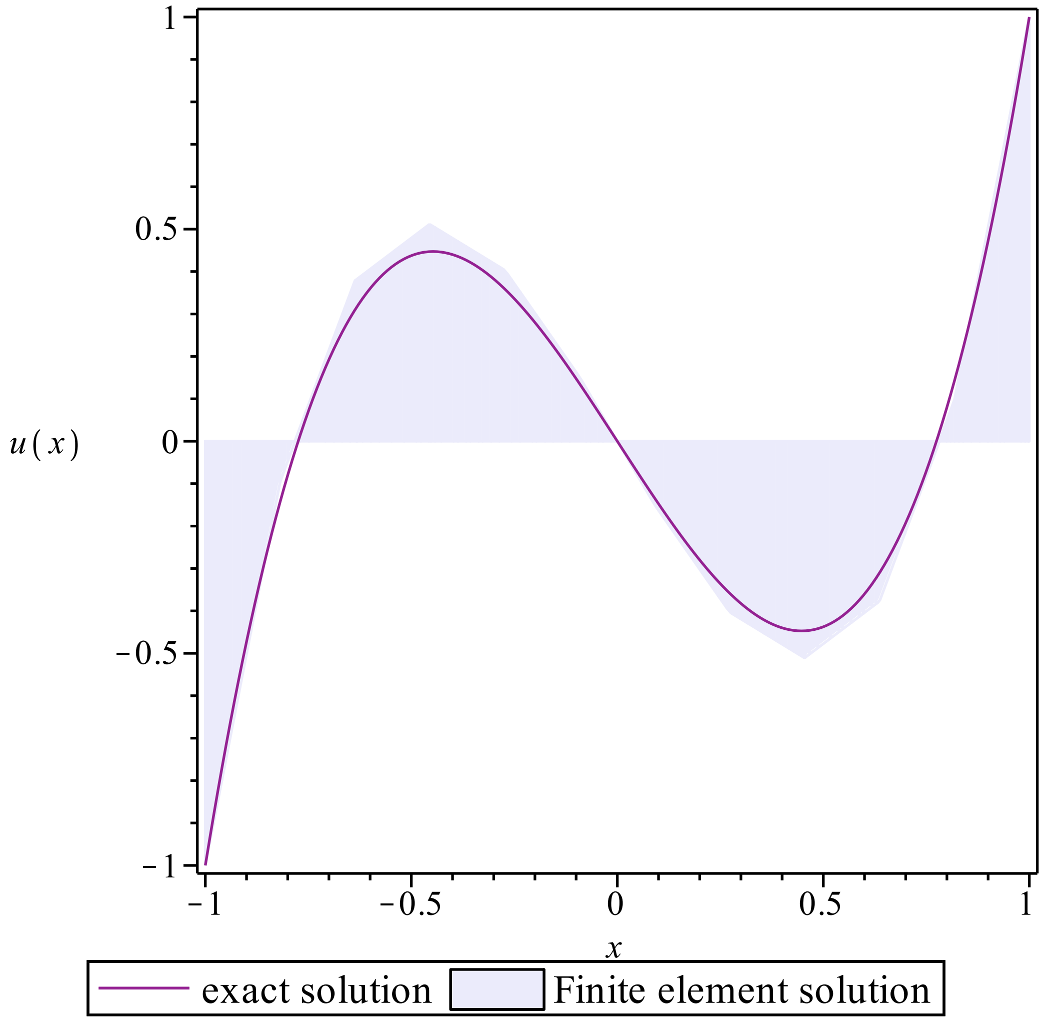
```
> l := 3:
P := x -> (1-x^2):
Q := x -> l*(l+1):
R := x -> 0:
alpha := (-1)^l;
beta := 1;
rho := x -> 0:
lambda := x -> 0:
ode := subs(u(t,x)=u(x),pde);
u_sol := subs(dsolve([ode,u(-1)=alpha,u(1)=beta]),u(x));
ans := EllipticSolver(10,P,Q,R,alpha,beta):
display([ans,plot(u_sol,x=-1..1,color="Purple",legend="exact solution")],axes=boxed,
labels=[x,u(x)]);
```

$$\alpha := -1$$

$$\beta := 1$$

$$ode := -\left(\frac{d}{dx}\left(\left(1-x^2\right)\left(\frac{d}{dx}u(x)\right)\right)\right) - 12\,u(x)$$

$$u\_sol := -\frac{3}{2}x + \frac{5}{2}x^3$$



Parabolic initial value problems

In this section, we will concentrate on case when $\rho(x) = 0$ in **(1.1)**; i.e., the equation we want to solve is:

$$\lambda(x)\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[P(x)\frac{\partial u}{\partial x}\right] + Q(x)u - R(x), \quad u(-1) = \alpha, \quad u(1) = \beta.$$

In this case, the matrix equation we need to solve for the amplitudes **a** of the basis functions is

$$C\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} + K\mathbf{a} = \mathbf{f}.$$

This can be re-arranged to give:

$$\frac{\mathrm{d}\mathbf{a}}{\mathrm{d}t} = A\mathbf{a} + \mathbf{g}, \quad A = -C^{-1}K, \quad \mathbf{g} = C^{-1}\mathbf{f}.$$

We solve this matrix ODE by introducing a time lattice $t_i = ih$, where h is the stepsize. We write $\mathbf{a}_i = \mathbf{a}(t_i)$ and use a trapezoidal stencil to generate the numeric solution:

$$\left(I - \frac{h}{2}A\right)\mathbf{a}_{i+1} = \left(I + \frac{h}{2}A\right)\mathbf{a}_i + h\mathbf{g}$$

Here is a procedure that implements this method assuming initial data of the form $u(0,x) = \kappa(x)$. **N1** is the number of interior finite elements, **N2** is the number of time steps, and we assume the simulation runs from **t=0** to **t=t_max**:

```
> ParabolicSolver := proc(N1,N2,kappa,t_max,lambda,P,Q,R,alpha,beta)
    local phi, X, T, a, K, C, f, Cinv, A, g, h, Gamma, p, i ,N:

    phi := GenerateBasis(N1):
    X := i -> -1 + 2*i/(N1+1):
    T := j -> j/N2*t_max:
    a := Vector([seq(evalf(kappa(X(i))),i=1..N1)],datatype=float):
    K := stiffness(phi,P,Q):
    C := damping(phi,lambda):
    f := force(phi,P,Q,R,alpha,beta):
    Cinv := C^(-1):
    A := -Cinv.K:
    g := Cinv.f:
    h := evalf(T(1)-T(0));
    Gamma[1] := 1 - h*A/2:
    Gamma[2] := 1 + h*A/2:
    p[0] := Frame(a,phi,alpha,beta,T(0)):
    for i from 1 to N2 do:
       a := LinearSolve(Gamma[1],Gamma[2].a+h*g):
```

```
        p[i] := Frame(a,phi,alpha,beta,T(i)):
     od:
     display(convert(p,list),insequence=true):

  end proc:

  Frame := proc(ans,Phi,alpha,beta,T) local N:
     N := rhs(ArrayDims(Phi))-1;
     plot([seq(alpha*Phi[0](x)+add(ans[i]*Phi[i](x),i=1..N)+beta*Phi[N+1](x),j=1..2)],x=-1.
  .1,filled=[true,false],color=["Lavender","Purple"],axes=boxed,title=typeset(t=evalf[4](T)
  ),labels=[x,u(t,x)]);
  end proc:
```

Here is an example of the output of `ParabolicSolver`:

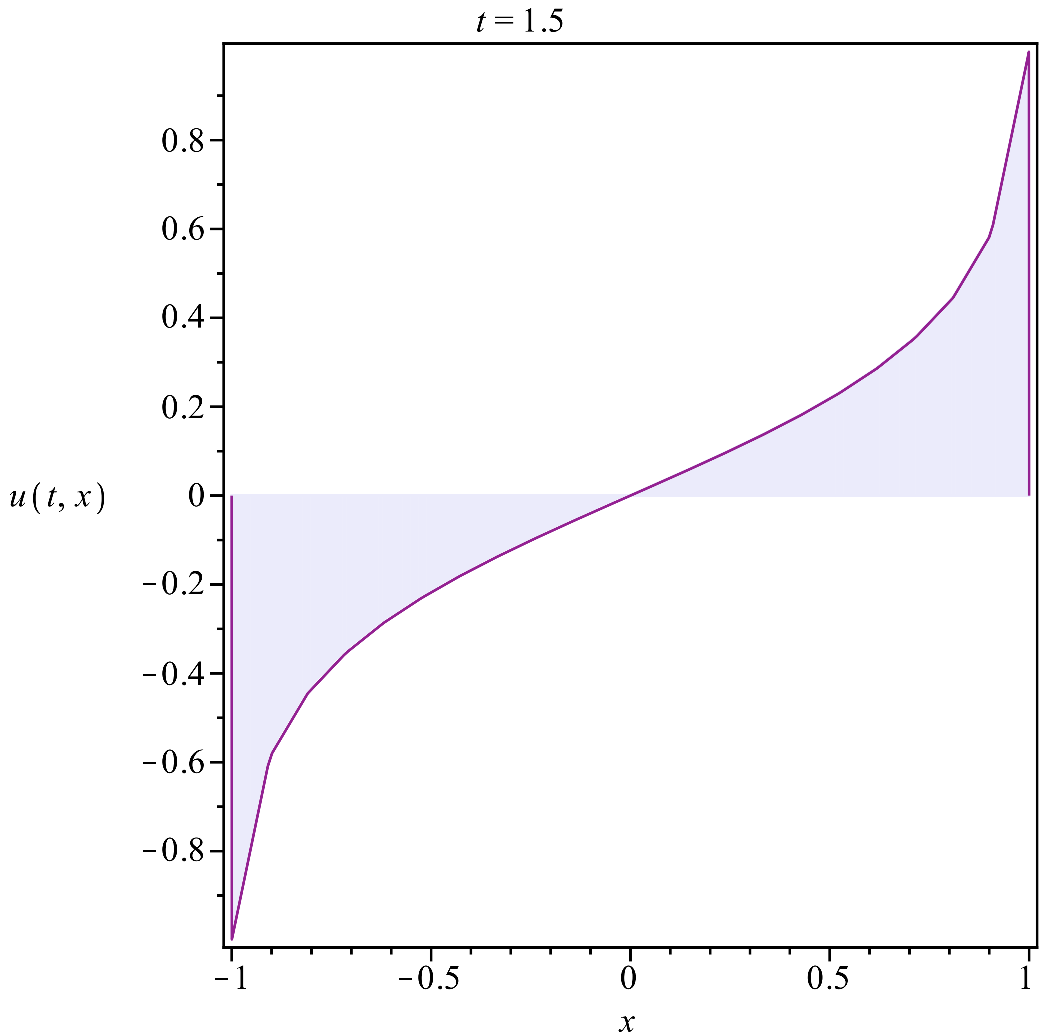
```
> P := x -> 1-x^2:
  Q := x -> 0:
  R := x -> 0:
  alpha := -1;
  beta := 1;
  rho := x -> 0:
  lambda := x -> 1:
  pde;
  kappa := x -> sin(5*Pi*x/2);
  N1 := 20:
  N2 := 100:
  t_max := 1.5:
  ParabolicSolver(N1,N2,kappa,t_max,lambda,P,Q,R,alpha,beta);
```

$$\alpha := -1$$

$$\beta := 1$$

$$\frac{\partial}{\partial t} u(t,x) - \left(\frac{\partial}{\partial x}\left(\left(1-x^2\right)\left(\frac{\partial}{\partial x} u(t,x)\right)\right)\right)$$

$$\kappa := x \rightarrow \sin\left(\frac{5}{2}\pi x\right)$$



Hyperbolic initial value problems

In this section, we concentrate on the complete PDE **(1.1)**:

$$\rho(x)\,\frac{\partial^2 u}{\partial t^2} + \lambda(x)\,\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left[P(x)\,\frac{\partial u}{\partial x}\right] + Q(x)u - R(x), \quad u(t,-1) = \alpha, \quad u(t,1) = \beta.$$

In this case, the matrix ODE to solve is

$$M\frac{d^2\mathbf{a}}{dt^2} + C\frac{d\mathbf{a}}{dt} + K\mathbf{a} = \mathbf{f}.$$

We convert this into a set of coupled first order ODEs via the definition $\mathbf{b} = \frac{d\mathbf{a}}{dt}$:

$$\frac{d\mathbf{a}}{dt} = \mathbf{b}, \qquad \frac{d\mathbf{b}}{dt} = -M^{-1}K\mathbf{a} - M^{-1}C\mathbf{b} + M^{-1}\mathbf{f},$$

This can in turn be represented by a single matrix ODE:

$$\frac{d\mathbf{B}}{dt} = A\mathbf{B} + \mathbf{g}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}, \quad A = \begin{bmatrix} 0 & I \\ -M^{-1}K & -M^{-1}C \end{bmatrix}, \quad \mathbf{g} = \begin{bmatrix} 0 \\ M^{-1}\mathbf{f} \end{bmatrix}.$$

We will solve this with the same type of trapezoidal stencil that we used in the last section; i.e., we write $\mathbf{B}_i = \mathbf{B}(t_i)$ with $t_{i+1} - t_i = h$ and use a trapezoidal stencil to generate the numeric solution:

$$\left(I - \frac{h}{2}A\right)\mathbf{B}_{i+1} = \left(I + \frac{h}{2}A\right)\mathbf{B}_i + h\mathbf{g}$$

Here is a procedure that implements this method assuming initial data of the form $u(0,x) = \kappa(x)$ and $u(0,x) = \xi(x)$. **N1** is the number o interior finite elements, **N2** is the number of time steps, and we assume the simulation runs from **t=0** to **t=t_max**:

```
> HyperbolicSolver := proc(N1,N2,kappa,xi,t_max,rho,lambda,P,Q,R,alpha,beta)
    local phi, X, T, a, K, C, f, Minv, A, g, h, Gamma, p, i ,N, b, B, M:

    phi := GenerateBasis(N1):
    X := i -> -1 + 2*i/(N1+1):
    T := j -> j/N2*t_max:
    a := Vector([seq(evalf(kappa(X(i))),i=1..N1)],datatype=float):
    b := Vector([seq(evalf(xi(X(i))),i=1..N1)],datatype=float):
    B := Vector([a,b]):
    K := stiffness(phi,P,Q):
```

```
    C := damping(phi,lambda):
    M := mass(phi,rho):
    f := force(phi,P,Q,R,alpha,beta):
    Minv := M^(-1):
    A := Matrix([[ZeroMatrix(N1,N1),IdentityMatrix(N1,N1)],[-Minv.K,-Minv.C]]):
    g := Vector([ZeroMatrix(N1,1),Minv.f]):
    h := evalf(T(1)-T(0));
    Gamma[1] := 1 - h*A/2:
    Gamma[2] := 1 + h*A/2:
    p[0] := Frame(B[1..N1],phi,alpha,beta,T(0)):
    for i from 1 to N2 do:
       B := LinearSolve(Gamma[1],Gamma[2].B+h*g):
       p[i] := Frame(B[1..N1],phi,alpha,beta,T(i)):
    od:
    display(convert(p,list),insequence=true):

 end proc:
```

Here is some example output:

```
> P := x -> 1-x^2:
  Q := x -> 0:
  R := x -> 0:
  alpha := -1;
  beta := 1;
  rho := x -> 1:
  lambda := x -> 0:
  pde;
  kappa := x -> sin(Pi*x/2);
  xi := x -> x^2;
  N1 := 40:
  N2 := 100:
  t_max := 20:
  HyperbolicSolver(N1,N2,kappa,xi,t_max,rho,lambda,P,Q,R,alpha,beta);
```

$$\alpha := -1$$

$$\beta := 1$$

$$\frac{\partial^2}{\partial t^2} u(t,x) - \left(\frac{\partial}{\partial x}\left(\left(1-x^2\right)\left(\frac{\partial}{\partial x} u(t,x)\right)\right)\right)$$

$$\kappa := x \to \sin\left(\frac{1}{2}\pi x\right)$$

$\xi := x \rightarrow x^2$