Bernstein Polynomials

A Bernstein polynomial of order N is defined as

$$x_N(t) = \sum_{i=0}^{N} \bar{x}_{i,N} b_{i,N}(t), \quad t \in [0, t_f]$$

where $\bar{x}_{i,N} \in \mathbb{R}$ is the Bernstein coefficient and $b_{i,N}(t)$ is the Bernstein polynomials basis defined as

$$b_{i,N}(t) = \binom{N}{i} \frac{t^i (t_f - t)^{N-i}}{t_f^n},$$

where

$$\binom{N}{i} = \frac{N!}{i!(N-i)!}$$

is the binomial coefficient. Notice that, for a given range of argument $[0, t_f]$, Bernstein polynomials are uniquely defined by the Bernstein coefficients: unlike Lagrange polynomials, Bernstein polynomials do NOT satisfy the Kronecker property and they are NOT interpolating polynomials. In what follows, we show an example of Bernstein polynomials for given Bernstein coefficients.

```
clear all; close all;
time = 0:0.01:7;
xbar = [1 4 2 3];
xN = BernsteinPoly(xbar,time);
figure
plot(time,xN,'Linewidth',3,'Color','r');
grid on
```

Bernstein Quadrature

Recall that numerical quadrature is an approximation method for definite integrals. Bernstein quadrature establishes weights w_i for given equadistant nodes $t_i = \frac{i}{N} t_f$ to approximate an integral as a finite sum:

$$\int_a^b f(t)dt \approx \sum_{i=1}^{N} w_i f(t_i)$$

Note that the above finite sum involves evaluation of the function $f(t)$ at the equidistant nodes $\{t_i\}_{i=0}^{N}$. The weights are given by

$$w_i = \frac{t_f}{N+1} \qquad \forall i \in \{0, \dots, N\}$$

Example

Define the following functions

$$u_1(t) = \frac{1}{\sqrt{1+(T-t)^2}}$$

$$u_2(t) = \frac{T-t}{\sqrt{1+(T-t)^2}}$$

with $T = 12$.

```
T = 12;
u1 = @(t) 1./(sqrt(1+(T-t).^2));
u2 = @(t) (T-t)./(sqrt(1+(T-t).^2));
```

We plot these functions for $t \in [0, T]$

```
t = 0:0.01:T;
u1fig = figure;
plot(t,u1(t),'LineWidth',3,'Color','k'); hold on
grid on
u2fig = figure;
plot(t,u2(t),'Linewidth',3,'Color','k'); hold on
grid on
```

The integrals of $u_1(t)$and $u_2(t)$ are computed analytically as

$$\int_0^T u_1(t)dt = \text{asinh}(T), \quad \int_0^T u_2(t)dt = \sqrt{T^2+1} - 1$$

which are defined on Matlab as follows

```
intu1 = asinh(T);
intu2 = sqrt(T^2+1)-1;
```

The following code computes the integral numerically using Bernstein quadrature. First, we define the order of approximation:

```
N = 80;
```

Then, we compute the integration weights and the equidistant nodes (though this should be quite straightforward, we use the BeBOT package)

```
[tnodes, w, D] = BeBOT(N,T);
```

Then, we evaluate the functions $u_1(t)$ and $u_2(t)$at the time nodes

```
u1_N = u1(tnodes);
figure(u1fig);
plot(tnodes,u1_N,'o','LineWidth',3,'Color','k'); hold on
grid on
u2_N = u2(tnodes);
figure(u2fig);
```

```
plot(tnodes,u2_N,'o','LineWidth',3,'Color','k'); hold on
grid on
```

Finally, we apply the quadrature rule

```
int_u1 = u1_N*w;
int_u2 = u2_N*w;
```

and compare the analytical solution with the numerical one

```
err1 = int_u1 - intu1
err2 = int_u2 - intu2
```

You are welcome to play with the order of approximation (Line 21) to verify how the integration error decreases when increasing N.

Convergence properties

The following results hold for Bernstein quadrature:

if $f(t) \in \mathscr{C}^0$ on $[0, t_f]$ then we have

$$\left\| \int_0^{t_f} f(t)dt - \sum_{j=0}^{N} w_j f(t_j) \right\| \leq C_I W_f(N^{-1/2})$$

where W_f is the modulus of continuity of $f(t)$ and $C_I > 0$ is independent of N. Moreover, if $f(t) \in \mathscr{C}^2$ then

$$\left\| \int_0^{t_f} f(t)dt - \sum_{j=0}^{N} w_j f(t_j) \right\| \leq \frac{C_I}{N}$$

Bernstein Approximation

Given any function $f(t)$, the Bernstein polynomial $f_N(t)$ of degree N with Bernstein coefficients given by the function computed at equidistant nodes is called the Bernstein approximation of $f(t)$. For points

$$t_i = \frac{i}{N} t_f$$

this can be written as

$$f_N(t) = \sum_{i=0}^{N} f(t_i) b_{i,N}(t)$$

Example

For the functions $u_1(t)$ and $u_2(t)$ defined earlier and with order of approximation and equidistant time nodes

```
N = 5;
[t_nodes, ~, ~] = BeBOT(N,T);
```

the Bernstein approximations are given by

```
u1_N = BernsteinPoly(u1(t_nodes),t);
u2_N = BernsteinPoly(u2(t_nodes),t);
```

The Bernstein approximation and the nodes are plotted here

```
figure
plot(t_nodes,u1(t_nodes),'o','Color','r','LineWidth',3); hold on;
plot(t,u1_N,'Color','r','LineWidth',3);
plot(t,u1(t),'Color','k','LineWidth',3);
grid on
figure
plot(t_nodes,u2(t_nodes),'o','Color','r','LineWidth',3); hold on;
plot(t,u2_N,'Color','r','LineWidth',3);
plot(t,u2(t),'Color','k','LineWidth',3);
grid on
```

Convergence properties

The following results hold for Bernstein approximation. Let $f(t) \in \mathscr{C}^0$ on $[0, t_f]$. Then, the Bernstein approximant satisfies

$$||f_N(t) - f(t)|| \leq C_0 W_f(N^{-1/2})$$

where $C_0 > 0$ is independent on N. Moreover, if $f(t) \in \mathscr{C}^2$ then

$$||f_N(t) - f(t)|| \leq C_0/N$$

Example: Non-smooth functions

The convergence of the Bernstein approximation is much slower than that of Lagrange interpolating polynomials. However, Bernstein approximation exhibits uniform convergence properties even when the function that is being approximated is non smooth (no Gibbs phenomenon).

Define the following function

$$u(t) = 1, \quad t \leq \bar{T}/2$$

$$u(t) = -1, \quad t > \bar{T}/2$$

for $\bar{T} = 2$. In Matlab we write

```
syms u(t)
Tbar = 2;
u(t) =@(t) piecewise(t<=Tbar/2, 1, Tbar/2<t, -1);
t = 0:0.01:Tbar;
uval = u(t);
```

which is plotted here

```
figure
plot(t,uval,'Linewidth',2,'Color','k'); hold on
grid on
```

Then, the Bernstein approximation of this function is computed and plotted as follows

```
N = 80;
[t_nodes,w,Dm] = BeBOT(N,Tbar);
u_N = BernsteinPoly(u(t_nodes),t);
plot(t_nodes,u(t_nodes),'o','Color','g','LineWidth',3);
plot(t,u_N,'Color','g','LineWidth',3);
```

You can modify the order of approximation (Line 52) to verify its dependence on the performance of the Bernstein approximant.

Bernstein Differentiation

Similarly to Lagrange interpolating polynomials, the Bernstein approximation of a function $f(t)$ by the Nth order Bernstein polynomial $f_N(t)$ also provides an estimate for the derivative $\dot{f}(t)$ through the derivative of $\dot{f}_N(t)$:

$$\dot{f}(t) \approx \dot{f}_N(t) = \sum_{i=0}^{N} f(t_i)\dot{b}_{i,N}(t)$$

The above equation can be written in matrix form as

$$\dot{f}_N(t) = \sum_{i=0}^{N}\sum_{j=0}^{N} f(t_j)D_{ji}b_{i,N}(t)$$

where D_{ji} is the (ij) entry of the differentiation matrix.

Consider the functions $u_1(t)$ and $u_2(t)$ defined earlier. Their derivatives are

$$\dot{u}_1(t) = \frac{T-t}{((T-t)^2+1)^{3/2}}$$

$$\dot{u}_2(t) = -\frac{1}{(T^2-2Tt+t^2+1)^{3/2}}$$

These are defined and plotted in Matlab as follows:

```
u1dot = @(t) (T-t)./((T-t).^2+1).^(3/2);
u2dot = @(t) -1./(T^2-2*T*t+t.^2+1).^(3/2);
t = 0:0.01:T;
fu1dot = figure;
plot(t,u1dot(t),'Linewidth',3,'Color','k'); hold on
grid on
fu2dot = figure;
plot(t,u2dot(t),'Linewidth',3,'Color','k'); hold on
grid on
```

These derivatives are then approximated by Bernstein polynomials as follows:

```
N = 50;
[t_nodes,w,Dm] = BeBOT(N,T);
u1dot_N = BernsteinPoly(u1(t_nodes)*Dm,t);
u2dot_N = BernsteinPoly(u2(t_nodes)*Dm,t);
figure(fu1dot);
plot(t_nodes,u1(t_nodes)*Dm,'o','Color','g','LineWidth',3); hold on
plot(t,u1dot_N,'Color','g','LineWidth',3);
figure(fu2dot);
plot(t_nodes,u2(t_nodes)*Dm,'o','Color','g','LineWidth',3); hold on
plot(t,u2dot_N,'Color','g','LineWidth',3);
```

Convergence properties

The following is true about the derivatives of Bernstein approximants. Let $f(t) \in \mathscr{C}^{r+2}$ on $[0, t_f]$, $r > 0$, and let $f_N(t)$ be the Bernstein approximant. Let $f^{(r)}(t)$ denote the rth derivative of $f(t)$. Then, the following inequalities hold:

$$||f_N(t) - f(t)|| \leq C_0/N$$

$$||\dot{f}_N(t) - \dot{f}(t)|| \leq C_1/N$$

$$\vdots$$

$$||f_N^r(t) - f^r(t)|| \leq C_r/N$$

Bernstein Integration

The integral of a Bernstein polynomial is also a Bernstein polynomial with control points computed as follows:

$$f(\bar{t}) = \dot{f} * I + f(0) * \mathbf{1}$$

where I is an integration matrix. In Matlab this can be computed as follows. We first define Bernstein coefficients

```
xn = [0 2 4 6 4 6 8 2 3 4 6];
```

describing a Bernstein polynomial of order

```
N = length(xn) - 1;
```

The derivative of this polynomial has control points

```
[t_nodes,w,Dm] = BeBOT(N,T);
xndot = xn*Dm;
```

If we integrate xndot we should recover xn. This is done as follows

```
I = BernsteinIntegrationMatrix(N,T);
xndot_int = xndot*I + ones(1,N+2)*xn(1);
```

We compare the results

```
figure
plot(t,BernsteinPoly(xn,t),'LineWidth',3); hold on
plot(t,BernsteinPoly(xndot_int,t),'LineWidth',3)
grid on
```