Taylor Polynomials: An Application of Derivatives to Approximating Functions

You may have noticed that operations with polynomials have many properties that make them popular to work with. This means that there are quite a few situations in the real world where approximating a more complicated function by a polynomial can result in a quick, good-enough solution. Let's investigate some of these approximations.

# Define a function and a point

Consider a function  and a point  in the domain of . Define your function as f(x) and your central point for the approximation as x0.

syms F(x) f(x)

F(x) = cos(x);

x0 = pi/2;

You defined:

fexp = F(x);

displayFormula("f(x) = fexp")



and

displayFormula("x\_0 = x0")



# Tangent Lines

You have probably encountered the first couple of steps in this process already. If we use a constant function, , the best approximation around the point  is simply to evaluate the function: . Visually, this looks like:

xval = x0-5:.1:x0+5;

clf

plot(xval,F(xval))

hold on

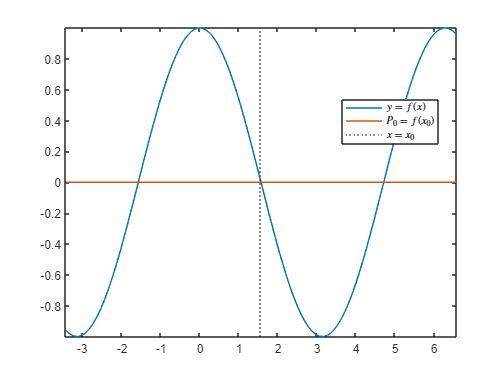
plot([xval(1) xval(end)],[F(x0) F(x0)])

xline(x0,":")

hold off

legend(["$y=f(x)$" "$P\_0=f(x\_0)$" "$x=x\_0$"],"Interpreter","latex","Location","best")

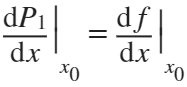
axis tight



Constant functions are usually pretty terrible approximations. We can get a significantly better approximation by considering the best linear approximation to  around . Theoretically, this means:



with

 and ,

or in other words

.

The best linear approximation to a differentiable function at a point is the tangent line at that point. In our example, this becomes:

syms dF(x) d f dx

dF(x) = diff(F(x),x);

dfexp = dF(x);

displayFormula("f(x) = fexp")



and

displayFormula("df/dx = dfexp")



so evaluating at  we have:

fx0 = F(x0);

dFx0 = dF(x0);

displayFormula("f(x0) = fx0")



and

displayFormula("df/dx = dFx0")



Therefore,  simplifies to

syms P(x)

P(x) = F(x0)+dF(x0)\*(x-x0);

displayFormula("P\_1 = fx0 + (dFx0)\*(x-x0)")



or, in slope-intercept form,

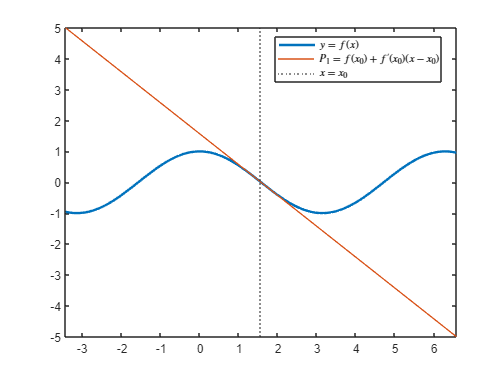
cval = fx0-dFx0\*x0;

displayFormula("P\_1 = dFx0\*x+cval")



approxLabel = "$P\_1 = f(x\_0)+f'(x\_0)(x-x\_0)$";

plotWithApprox(xval,x0,@(x)F(x),@(x)P(x),approxLabel)



## Errors

With the constant approximation, at the point  the constant approximation has an error of

.

With

x1 = vpa( x0+0.5);

x1show = double(round(x1,4));

this error becomes

fError = double(vpa(F(x1)-F(x0)));

disp("f("+x1show+") - P\_0("+x1show+") = " + fError)

f(2.0708) - P\_0(2.0708) = -0.47943

When the tangent line exists, however, it has an error of







which is guaranteed to be small for values of  close to . In this case, the error is

disp("f("+x1show+") - P\_0("+x1show+") = " + fError)

### Exercise 1: Tangent Lines

Given a function,  and a point , compute the tangent line to  at the point . Your solution should be .

clear

syms F(x) dF(x) d P(x) Psoln(x) x

num = randi([-10,10]);

idx = randi(7);

opts = [1 1 1 1 2 3 4];

x0 = num/opts(idx);

[myFun,varChoice,~] = genFun(x,1); % genProb is defined in Helper Functions

F(x) = myFun;

dF(x) = diff(F(x),x);

while imag(double(F(x0))) ~= 0 || imag(double(dF(x0))) ~= 0

[myFun,varChoice,~] = genFun(x,1); % genProb is defined in Helper Functions

F(x) = myFun;

dF(x) = diff(F(x),x);

end

displayFormula("f(varChoice) = myFun")

if mod(double(F(pi)),1) == 0

x0 = x0\*pi;

end

xVal = x0-5:0.1:x0+5;

displayFormula("x\_0 = x0")

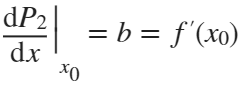
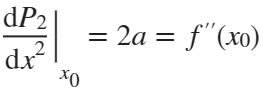
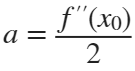
Psoln(x) = F(x0)+dF(x0)\*(x-x0);

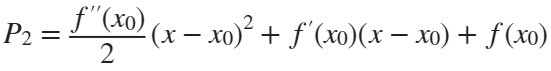
P(x) = log(40)+7/8\*(x-8);

plotWithApprox(xVal,x0,@(x)F(x),@(x)P(x),"My tangent line, $P(x)$")

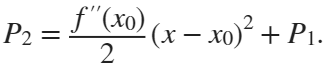
checkPofx(@(x)P(x),@(x)Psoln(x),x0)

# Quadratic Approximations

Why stop with linear approximations? Let's compute the best quadratic approximation to our function  around the point . Consider a general quadratic function, . Then  implies that . We also have the same condition that . The final condition is that . Therefore, , and the quadratic approximation to  at  is



which is also



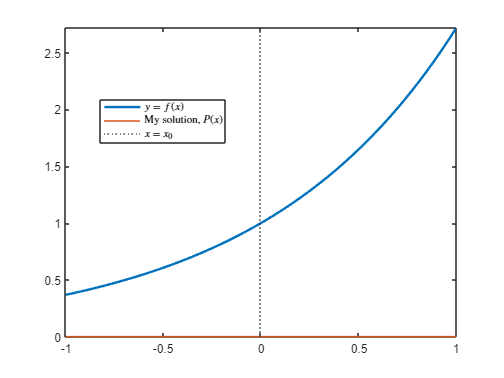
 **Exercise 2: Quadratic Approximation**

Find the quadratic approximation to  around the point . Your approximation should be .

syms P(x)

P(x) = 0;

plotWithApprox(-1:0.02:1,0,@(x)exp(x),@(x)P(x),"My solution, $P(x)$")



checkPPofx(@(x)P(x))

You have an incorrect value for f(0)

You have an incorrect value for f'(0)

You have an incorrect value for the quadratic term, a.

Your solution for the quadratic is incorrect. Please try again.

# Higher Order Approximation

We can continue to extend these calculations to higher-order approximations. A fifth-degree approximation, for instance, would be:

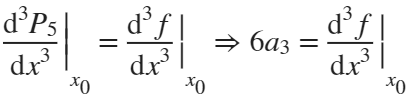


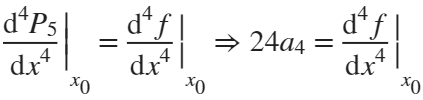
and  must agree with the function  we are trying to approximate and as many derivatives as possible.

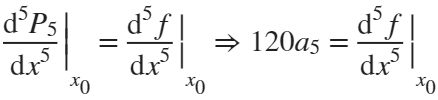












 **Exercise 3: Higher Order Approximation**

a) Calculate the fifth-degree approximation  for  around .

syms P5f(x)

P5f(x) = x-x^3/6+x^5/120;

b) Calculate the fourth-degree approximation  for  around .

syms P4g(x)

P4g(x) = 12-8\*(x+1)+3\*(x+1)^2;

c) Calculate the third-degree approximation  for  around .

syms P3h(x)

P3h(x) = 0+(x-1)-1/2\*(x-1)^2+1/3\*(x-1)^3;

checkHigherOrderApprox(@(x)P5f(x),@(x)P4g(x),@(x)P3h(x))

Your solution for part a) is correct.

Your solution for part b) is correct.

Your solution for part c) is correct.

 **Additional Practice: Create and Solve Your Own Problems**

Set up the problem:

syms x

f(x) = log(x);

x0 = 1;

n = 4;

Solve the problem of finding  for  around a center .

Pn(x) = (x-1)-(x-1)^2/2+(x-1)^3/3-(x-1)^4/4;

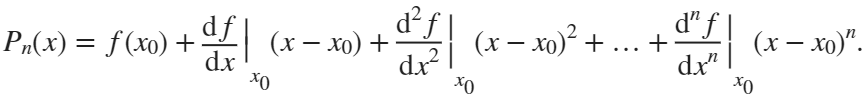
checkMyApprox(@(x)f(x),x0,n,@(x)Pn(x));

You have correctly solved the problem.

ans = 1

# Taylor Polynomials

Consider the  Taylor polynomial, , defined as:



We can visualize the resulting approximation:

syms F(x)

F(x) = exp(x);

x0 = 0;

deg = 4;

bds = [-pi,pi];

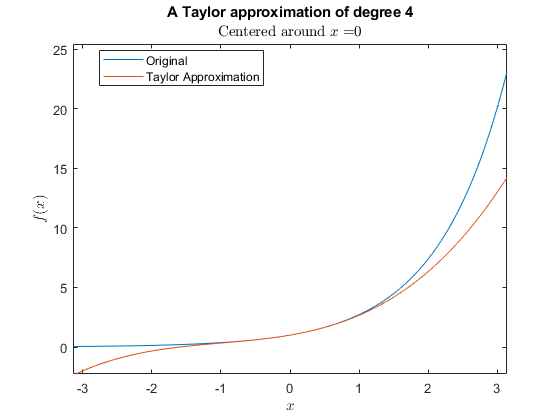
if deg<0 || mod(deg,1) ~= 0

disp("Please select a non-negative integer degree, n.")

else

T = showTaylor(@(x)F(x),deg,bds,x0);

end

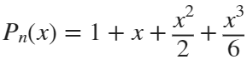


oldPrefs = sympref(); % Save the current user preferences to reset at the end of the section

sympref('PolynomialDisplayStyle','ascend'); % Present polynomials in descending power format

sympref('FloatingPointOutput',false); % Do not use floating point representations

displayFormula("P\_n(x) = T")



sympref(oldPrefs); % Restore the system preferences you had before running this script.

 **Reflect**.

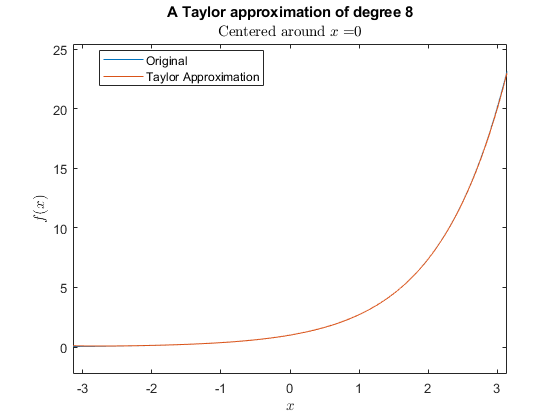
1. Investigate the Taylor polynomial approximations for  and . Do you have a theory as to why  is an odd function and  is an even function?

## Animate Approximation by Taylor Polynomials

Visualize the improvement in the approximation as the order of the polynomial used grows.

finalValue = 8;

Tlast = showTaylor(@(x)F(x),1:finalValue,bds,x0);



showT = false;

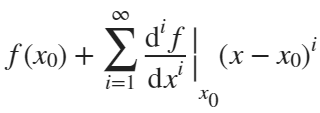
if showT

displayFormula("P\_n(x) = Tlast",["n","Tlast"],[finalValue,Tlast])

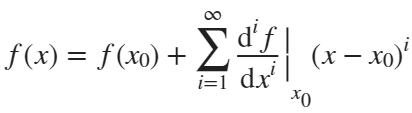
end

# Further Exploration: [Taylor Series](https://www.mathworks.com/help/symbolic/taylor-series.html)

If we take this to the limit and allow an infinite number of terms in our sum, we have the expression:



In fact, for sufficiently nice functions, called *analytic functions*, this expression converges to the value of the function itself:



# Local Helper Functions

plotWithApprox(xval,x0,Fh,dFh,approxLabel)

Create a plot of the original function over the domain given by xval. Layer on top of that a plot of  over the same domain. The final argument, approxLabel, provides the label used in the legend for the function .

function plotWithApprox(xval,x0,Fh,dFh,approxLabel)

clf

plot(xval,double(Fh(xval)),"LineWidth",2)

hold on

plot(xval,double(dFh(xval)))

xline(x0,":")

hold off

legend(["$y=f(x)$" approxLabel "$x=x\_0$"],"Interpreter","latex","Location","best")

axis tight

end

checkPofx(Ph,Psolnh,x0)

This function checks whether the expressions Ph and Psolnh match the way they are supposed to. It displays 'partial credit' comments if some of the computation matches, as well as noting the correctness or incorrectness of the entire solution.

function checkPofx(Ph,Psolnh,x0)

syms x

flag = [0 0];

if abs(double(Ph(x0) - Psolnh(x0))) < 4\*eps

disp("You have the correct value for f("+x0+")")

flag(1) = 1;

else

disp("You have an incorrect value for f("+x0+")")

end

if abs(double(diff(Ph(x),x)-diff(Psolnh(x),x)))<4\*eps

disp("You have the correct value for f'("+x0+")")

flag(2) = 1;

else

disp("You have an incorrect value for f'("+x0+")")

end

flag = logical(flag);

if all(flag)

disp("Your solution for the tangent line is correct.")

else

disp("Your solution for the tangent line is incorrect. Please try again.")

end

end

checkPPofx(Ph)

A modified version of checkPofx that checks the solutions offered to Exercise 2.

function checkPPofx(Ph)

syms x

x0 = 0;

flag = [0 0 0];

if abs(double(Ph(x0) -1)) < 4\*eps

disp("You have the correct value for f("+x0+")")

flag(1) = 1;

else

disp("You have an incorrect value for f("+x0+")")

end

if abs(double(subs(diff(Ph(x),x)-1,x,0)))<4\*eps

disp("You have the correct value for f'("+x0+")")

flag(2) = 1;

else

disp("You have an incorrect value for f'("+x0+")")

end

if abs(double(diff(Ph(x),2)-1))<4\*eps

disp("You have the correct value for the quadratic term, a.")

flag(3) = 1;

elseif abs(double(diff(Ph(x),x,2)/2-1))<4\*eps

disp("You have an incorrect value for the quadratic term, a.")

disp("Perhaps you forgot to divide by 2?")

else

disp("You have an incorrect value for the quadratic term, a.")

end

flag = logical(flag);

if all(flag)

disp("Your solution for the quadratic approximation is correct.")

else

disp("Your solution for the quadratic is incorrect. Please try again.")

end

end

checkHigherOrderApprox(P1,P2,P3)

This function checks that the solutions to Exercise 3 are correct by calling a helper function that checks each derivative.

function checkHigherOrderApprox(P1,P2,P3)

checkDers(P1,[0 1 0 -1 0 1],0, "a)");

checkDers(P2,[12 -8 6 0 0],-1,"b)");

checkDers(P3,[0 1 -1 2],1,"c)");

end

checkMyApprox(f,x0,n,Pn)

This function computes the appropriate arguments to pass into checkDers to evaluate whether the solution to the self-defined Additional Practice is correct.

function checkMyApprox(f,x0,n,Pn)

syms x

pSoln = nan([1 n+1]);

pSoln(1) = double(f(x0));

for k = 1:n

dkf = diff(f(x),x,k);

pSoln(k+1) = subs(dkf,x0);

end

checkDers(Pn,pSoln,x0,"extra")

end

checkDers(P,pSoln,x0,partLabel)

This function checks the derivatives of the proposed solution P against expected values defined by pSoln at the point x0. The input value partLabel is used to identify which Exercise component is currently being checked so feedback can be provided correctly.

function flag = checkDers(P,pSoln,x0,partLabel)

syms x

flag = 1;

p0 = pSoln(1);

pRest = pSoln(2:end);

if P(x0) ~= p0

flag = 0;

else

for k = 1:length(pRest)

dkP = diff(P(x),x,k);

if abs(double(subs(dkP,x0))-pRest(k)) > 4\*eps && flag == 1

flag = 0;

end

end

end

if partLabel ~= "extra"

if flag == 1

disp("Your solution for part " + partLabel + " is correct.")

else

disp("Your solution for part " + partLabel + " is incorrect.")

disp("Please try " + partLabel + " again.")

end

else

if flag == 1

disp("You have correctly solved the problem.")

else

disp("No, that is not the correct solution. Please try again.")

disp("You probably want to check your derivatives and your signs.")

end

end

end