9: Integration

9.1 Single Variable Integration

9.1.1 Symbolic Integration

If the Symbolic Toolbox is available, Matlab can calculate exact values of definite or indefinite integrals, as you learned to do in calculus II. For example, suppose we want to find $\int x\sin x\,dx$ (i.e. the antiderivative) and $\int_0^{\pi/4} x\sin x\,dx$. We first declare x a symbolic variable.

```
syms x
```

Example 9.1: (Indefinite Integral) Find $\int x\sin x\,dx$

Solution:

```
int(x*sin(x))

ans =

sin(x)-x*cos(x)
```

Matlab returns a specific antiderivative, without a constant of integration. ■

Example 9.2: (Definite Integral) Find $\int_0^{\pi/4} x\sin x\,dx$.

Solution:

```
int(x*sin(x),0,pi/4)

ans =

1/2*2^(1/2)-1/8*pi*2^(1/2)
```

In more traditional form we would write this as $\frac{1}{2}\sqrt{2}-\frac{1}{8}\pi\sqrt{2}$. If we wish a **numerical value** we can use the `double` command.

```
double(ans)

ans =

    0.1517
```

■

If you defined a symbolic function f (either anonymously or in an M-function) and you wish to find $\int f(x)dx$ you do not have to retype the expression $f(x)$ inside the `int` command. Just enter `int(f(x))`. For example, we have

```
f=@(x)x*sin(x);

int(f(x))
```

yielding

```
ans =

sin(x)-x*cos(x)
```

9.1.2 Numerical Integration

Matlab's ability to perform single variable symbolic integration is impressive. It can obtain answers to integration problems that are beyond the scope of elementary calculus.

For example, the result of

```
int(1/sqrt(1+x^2))

ans =

asinh(x)
```

may at least be familiar to you (it is the inverse hyperbolic sine function). But the answer to

```
int(1/sqrt(1+x^4))

ans =

1/(1/2*2^(1/2)+1/2*i*2^(1/2))*(1-
i*x^2)^(1/2)*(1+i*x^2)^(1/2)/(1+x^4)^(1/2)*EllipticF(x*(1/2*2^(1/2)+1/2*
i*2^(1/2)),i)
```

is certainly baffling, as it involves complex numbers (the *i* terms), as well as a new function `EllipticF`. There are a host of such special functions which arise in many advanced analytical problems. They are built into Matlab, much as sine and cosine, and Matlab uses them when theory indicates they are relevant.

Of course, even these special functions may fail to provide an answer to a problem. For example,

Example 9.3: Use Matlab to evaluate $\int \sqrt{1+\log x}\, dx$.

Solution:

```
int(sqrt(1+log(x)))

Warning: Explicit integral could not be found.

> In sym.int at 58

ans =

int((1+log(x))^(1/2),x)
```
■

In this case, Matlab gives up and issues a warning that it could not solve your problem. What do you do if you want to compute $\int_1^2 \sqrt{1+\log x}\, dx$? (Recall that in Matlab the natural logarithm is written as $\log$, not ln.)

Fortunately, Matlab has a very precise numerical integration routine based on Simpson's Rule, which you learned about in calculus II. We won't go into the details of the method. The command is called `quad`, which is short for quadrature, a somewhat obsolete term used for integration. The `quad` command works slightly differently than `int`. Instead of entering the expression that you want to integrate, it expects that you enter a handle to a function defining the expression. Moreover, your function needs to be vectorized, i.e. it must use "dot" operations or built-in functions such as `sin`, `sqrt`, etc, that are automatically vectorized.

Example 9.4: Use the `quad` command to evaluate $\int_0^{\pi/4} x \sin x \, dx$.

Solution:

```
quad(@(x)x.*sin(x),0,pi/4)


ans =

    0.1517
```

Of course, this agrees with the earlier result found using `int`. ■

In this example, we entered the function anonymously into the command. We can also first define the function using the @ construction. In that case, we simply pass the name of the function to the `quad` command (not the value $f(x)$ as we did for `int`). We can now numerically integrate the example that had stymied the symbolic processor.

Example 9.5: Use the `quad` command to evaluate $\int_1^2 \sqrt{1+\log x}\, dx$.

Solution:

```
g=@(x)sqrt(1+log(x));
quad(g,1,2)
ans =

    1.1743
```
■

In general, we would expect Matlab's numerical answer to be reliable. Nonetheless, it is a good idea to check things when you can do so. For example, we can try to perform a symbolic integration and, assuming Matlab can carry it out, compute a numerical approximation from the latter. If there are discrepancies, we will have to investigate more deeply to figure out which answer, if any, is correct. This may not be easy, but one way to do so is to write your own simple routine that gives an approximate answer. In the case of integration, we can use the simple scheme of mid-point Riemann sums. This works as follows

Mid-point Riemann Sums to approximate $\int_a^b f(x)\,dx$,

a) Select a whole number n.
b) Divide the interval of integration $[a,b]$ into n equal subintervals each of length $dx = \frac{b-a}{n}$.

c) Compute the mid-point of each subinterval. These are the n points $x_1 = a + \frac{dx}{2}$, $x_2 = x_1 + dx$, $\ldots, x_n = x_{n-1} + dx$. The last point will be $b - \frac{dx}{2}$.

d) Compute the Riemann Sum

$$f(x_1)dx + f(x_2)dx + \cdots + f(x_n)dx = (f(x_1) + f(x_2) + \cdots + f(x_n))dx$$

e) The latter sum will approach $\int_a^b f(x)\,dx$ as n increases.

It is very easy to implement these steps as an M-file. We do this for the function $f(x) = \sqrt{1 + \log x}$. The following code has been saved as the M-file `midpoint.m` and the reader can modify it for use with a different example.

Example 9.6: Compute the Mid-Point Riemann Sum Approximation to $\int_1^2 \sqrt{1 + \log x}dx$

Solution:

```
f=@(x)sqrt(1+log(x));  % function should be vectorized
a=1; b=2;  % interval of integration
n=20;  % number of subintervals.
% The rest of the code just implements the method.
dx=(b-a)/n;
x=a+dx/2:dx:b-dx/2;  % generate the midpoints
y=f(x);  % compute the function values at the midpoints
RS=sum(y)*dx  % sum command adds the entries in the vector y
```

```
   RS =

     1.1744
```

■

This is in sufficient agreement with the earlier result for us to accept the latter. If we had selected a larger number of subintervals we would have, in theory, seen closer agreement. In practice, using an extremely large a number of subintervals may introduce rounding errors that will degrade the theoretical accuracy of the method. This is more of a concern in multivariate integration.

9.2 Multiple Integration

9.2.1 Symbolic Double Integration

The `int` command can be nested to carry out iterated symbolic double integration. Matlab's graphing facilities may be able to assist you in visualizing the region of integration in order to set up the double integral. However, we assume at this point that you have formulated an iterated double integral to solve your problem. Let's do some examples illustrating how to do the integration. First we declare both x and y symbolic variables.

```
syms x y
```

Example 9.7: Compute $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ **(Integration over a rectangle)**

Solution: To avoid excessive parentheses, we prefer to do the inner integral separately and then pass the result to the outer integral.

```
inner=int(x^2+y^2,x,0,2) % we must specify the variable of integration x

int(inner,0,1) % the variable of integration now defaults to the
remaining variable.

inner =

8/3+2*y^2

ans =

10/3
```

We could have suppressed the display of the inner integral. For those who prefer, the entire calculation can be entered in one line as

```
int(int(x^2+y^2,x,0,2),0,1)

ans =

10/3
```

■

If the region of integration is not a rectangle, we may enter appropriate formulas for the boundaries.

Example 9.8: Evaluate $\int_0^1 \int_0^x x^2 + y^2 dydx$

Solution:

```
inner=int(x^2+y^2,y,0,x); int(inner,0,1)

ans =

1/3
```

■

9.2.2 Numeric Double Integration

It is much more common that a symbolic integration package will fail to evaluate a double integral because of the increased complexity arising after the initial integration. It is therefore desirable to have a method for approximating such integrals numerically. Matlab has a procedure `dblquad` for doing this. This procedure works well for rectangular regions of integration, provided the integrand is continuous on the domain, including along the boundary. With some tweaking it is possible to use it on non-rectangular domains, but the results are often unreliable. We therefore do not recommend its use. In the next section we describe an alternative procedure called `quadvec`, which is not part of the Matlab system but has been developed by the Mathworks, works quite well, and can easily be extended to do numeric triple and higher order integrations, at least in principle.

As with `quad`, the `dblquad` command expects its function input to be the handle of a Matlab function. Moreover, this function must accept vectorized inputs.

Example 9.9: Evaluate $\int_0^1\int_0^2 x^2 + y^2 dxdy$ using `dblquad.`

Solution:

```
f=@(x,y)x.^2+y.^2;  %define anonymous function.  Note the . operations.
dblquad(f,0,2,0,1)  % first give x-limits, then y-limits


ans =

    3.3333
```

To the displayed precision, this agrees with the exact answer 10/3 computed above. ■

We can also do a quick check on the accuracy of this result by creating our own Riemann Sum approximation. This is similar to what we did above in the single variable case. We generate a 2D grid of midpoints and evaluate the function at each grid point. These values are summed and the result multiplied by $dx\,dy$ to produce a Riemann Sum approximation to the double integral. The only unexpected revision in the code is that to sum all the entries in a 2D array (matrix) we have to apply the `sum` command twice. On a 2D array, the `sum` command simply adds all entries in each column. We then apply the command again to add those entries. For example,

```
A=[1 2; 3 4]  % enters a 2 x 2 matrix

A =

     1     2

     3     4

sum(A)  % sums entries in each column

ans =

     4     6

sum(ans)  % sums the entries in the last vector, thus giving total for A

ans =

    10

sum(sum(A))  % this sums all entries in A.  Notice the similarity to
using a double integral.

ans =

    10
```

Example 9.10: Construct an M-file to find a mid-point Riemann sum approximation to a double integral (over a rectangle). (This file is available as `dblmidpoint.m`.)

Solution:

```
f=@(x,y)x.^2+y.^2;  % function should be vectorized
```

```
a=0; b=2;  % x axis limits of integration
c=0; d=1;  % y axis limits of integration
m=20;  % number of x-subintervals.
n=20; % number of y-subintervals
% The rest of the code implements the method.
dx=(b-a)/m; dy=(d-c)/n;
x=a+dx/2:dx:b-dx/2;  % generate the x-midpoints
y=c+dy/2:dy:d-dy/2;  % generate the y-midpoints
[x y]=meshgrid(x,y); % 2D array of grid coordinates of midpoints
z=f(x,y);  % compute the function values at the midpoints
RS = sum(sum(z))*dx*dy


RS =
    3.3312
```

The answer only agrees to the second decimal place with the more precise value using `dblquad`. Using a larger number of subintervals will produce closer agreement. ■

9.2.3 Alternative Numeric Double Integration

The `dblquad` command is awkward to use with non-rectangular domains and often suffers from a lack of accuracy on such domains. One might think that nested `quad` commands could be used to numerically evaluate a double integral, much as nested `int` commands can be used to symbolically evaluate double integrals. Let's see what happens when we try to carry out such a plan. We consider $\int_0^1\int_0^2 x^2+y^2dxdy$. First we define a numeric inner integral by entering

```
inner=@(y)quad(@(x)x.^2+y.^2,0,2)  % computes a function that gives the
numeric value of the inner integral for any value y of outer integration


inner =
    @(y)quad(@(x)x.^2+y.^2,0,2)
```

We can evaluate such a function in the usual way. For example,

```
inner(1.5)


ans =
    7.1667
```

This is just $\int_0^2 (x^2 + (1.5)^2)dx$. The `inner` function passes the value $y = 1.5$ to the integrand in `quad` and then `quad` performs a numerical integration. Note, however, that the function `inner` is not vectorized. If we ask

```
inner([1, 1.5])
```

```
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> @(x)x.^2+y.^2
Error in ==> quad at 63
y = f(x, varargin{:});
Error in ==> @(y)quad(@(x)x.^2+y.^2,0,2)
```

we get a series of error messages. Why is this important? Because, if we want to integrate the `inner` function using `quad` (to complete the double integral) we must supply the function handle of a vectorized function and `inner` is not vectorized. Thus, writing

```
quad(inner, 0,1)
```

```
??? Error using ==> plus
Matrix dimensions must agree.
Error in ==> @(x)x.^2+y.^2
Error in ==> quad>quadstep at 118
y = f(x, varargin{:});
Error in ==> quad at 78
...
```

produces a similar collection of error messages.

The M-function `quadvec` uses the `quad` function (actually a more precise version known as `quadl`), but automatically vectorizes the input function you give it. This allows us to plug in function handles of non-vectorized functions and `quadvec` will vectorize them before passing them to the `quad` function. Assuming `quadvec` is in the path accessed by Matlab we can then compute $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ as follows.

Example 9.11: Compute the value of $\int_0^1 \int_0^2 x^2 + y^2 dxdy$ using the routine `quadvec`.

Solution:

```
inner=@(y)quadvec(@(x)x^2+y^2,0,2);
quadvec(inner,0,1)
```

```
ans =

    3.3333
```

Notice that we did not have to vectorize even the initial function expressions, because `quadvec` automatically takes care of that. Naturally, we can write the entire expression as a nested function, though the parentheses get rather thick. The @ designations in effect serve to indicate the variable of integration in each part.

```
quadvec(@(y)quadvec(@(x)x^2+y^2,0,2),0,1)


ans =

    3.3333
```

■

One additional advantage of this command is that it allows us to easily carry out numeric integrations over non-rectangular regions.

Example 9.12: Evaluate $\int_0^1 \int_0^{\sqrt{1-x^2}} x^2 + y^2 dydx$ numerically using `quadvec`.

Solution:

```
inner = @(x)quadvec(@(y)x^2+y^2,0,sqrt(1-x^2));

quadvec(inner,0,1)


Warning: Minimum step size reached; singularity possible.
> In quadl at 95
  In quadvec at 22
  In @(x)quadvec(@(y)x^2+y^2,0,sqrt(1-x^2))
  In quadvec>g at 26
  In quadl at 64
  In quadvec at 22
ans =

    0.3927
```

Although the command issued a number of warnings, it did produce the correct numerical value, (to the default tolerance of 10^{-6}), as we can verify in this case using symbolic integration:

```
syms x y;

int(int(x^2+y^2,y,0,sqrt(1-x^2)),0,1);

double(ans)


ans =

    0.3927
```

■

9.2.4 Triple Integrals

Much of what we have said extends to triple integrals. We can nest the `int` function to duplicate the computation of iterated triple integrals. Similarly we can iterate `quadvec` to handle examples that can only be done numerically. We demonstrate both methods by computing a common integral. We do not recommend the built-in routine `triplequad`, unless you need to compute a triple integral over a rectangular box with sides parallel to the coordinate planes.

Example 9.13: Find the exact value of $\int_0^1\int_0^z\int_0^y ze^{-y^2}\,dxdydz$ using the `int` command.

Solution: We need to create three symbolic variables. Then we use `int` to carry out the successive integrations. We show the intermediate steps so the reader can compare the computation with the standard evaluation done by hand.

```
syms x y z
intx=int(z*exp(-y^2),x,0,y)
inty=int(intx, y, 0, z)
int(inty, 0, 1)
double(ans)  % convert to numeric answer
```

```
intx =
z*exp(-y^2)*y
inty =
-1/2*z*exp(-z^2)+1/2*z
ans =
1/4*exp(-1)
ans =

    0.0920
```
■

Now we repeat the calculation using `quadvec`. The steps are very similar to those displayed above, except we pass function handles at each call of the `quadvec` function.

Example 9.14: Approximate the value of $\int_0^1\int_0^z\int_0^y ze^{-y^2}\,dxdydz$ using the `quadvec` command.

Solution:

The computation generates many warnings, similar to those displayed above in Example 9.12. In order not to fill the page with these warnings, we suppress that part of the output.

```
warning off all;  %turn off warnings
f=@(x,y,z)z*exp(-y^2);
intyz=@(y,z)quadvec(@(x)f(x,y,z),0,y);
intz=@(z)quadvec(@(y)intyz(y,z), 0, z);
quadvec(intz,0,1)
warning on  % turn on warnings at exit
```

```
ans =

    0.0920
```

This agrees with the result obtained earlier using symbolic iterated triple integrals.

Notice that whenever the integrand involves more than one variable, we have to create a function handle from this integrand that only involves the single variable that we wish to integrate. Thus, the @*variable* symbol at each stage serves as a reminder of the corresponding d*variable* in the traditional integral. We could have done this at the end as well, writing `quadvec(@(z)intz(z),0,1)`, though it is superfluous at that stage.■

9.3 Summary

We discussed both numerical and symbolic single and multiple integrals. We used the following built-in Matlab functions:

`int`: computes symbolic integrals; can be nested to do multiple integrals. Input must be a symbolic expression.

`double`: converts an exact (symbolic) value to double precision floating point (decimal).

`quad`: single variable numeric integration. Input must be the handle to a vectorized function of a single variable. Cannot be nested to do multiple numeric integration.

`dblquad` : double integral numeric integration. Input must be the handle to a vectorized function of two variables. Cannot be applied directly to non-rectangular domains. See also **`triplequad`** for triple integrals over 3D boxes.

In addition, we discussed several M-files as enhancements or checks on the system files.

`midpoint`: implements midpoint Riemann sum approximation to a single integral

`dblmidpoint`: implements midpoint Riemann sum approximation to a double integral over a rectangle.

`quadvec`: vectorizing numerical integrator. Can be used for multivariate numeric integration.

9.4 Exercises

Exercise 9.1: Let R be the region in the plane bounded by $y = x^2$, $y = 0$, and $x = 1$. Set up an iterated double integral to evaluate $\iint_R e^{y/x}\, dA$.

Your work should produce a single M-file that gives the following output:

a) Uses Matlab's symbolics to find the exact value of the double integral.

b) Uses `quadvec` to obtain a numerical estimate for the double integral.

Publish your work to html.

Note: In publishing your work, insert cell dividers after each output that you want displayed. This way the output will appear directly after the input that generates it, rather than having all the output at the end, where one can't tell what it goes with. See the Cell menu for the command to enter a cell divider.

Exercise 9.2: Using Matlab write an M-file that does the following:

a) Draws the region *R* between the curves $y = (1 - x^{2/3})^{3/2}$ and $y = 1 - x$ for $0 \le x \le 1$.

b) Assuming the region *R* (in 9.2a) is a plane lamina of density 1, finds its exact mass using symbolic double integrals. Convert to a numerical value using `double`.

c) Use the `quadvec` command to confirm the answer in b) by numerical calculation.

d) Uses Matlab to find the exact expressions for the moments of the region *R* about the *x* and *y* axes.

e) Using d) and either the answer to b) or c) find a numerical approximation to the coordinates of the center of mass of *R*. (N.B. The symmetry of the region suggests that you should find $\bar{x} = \bar{y}$.)

Publish your M-file in html. Make sure you indicate by suitable comments what you are calculating in each part. Also, see the note in Exercise 9.1 concerning separation of the output when you publish the M-file.

Exercise 9.3 A solid hemisphere defined by $0 \le x^2 + y^2 + z^2 \le 1$ and $z \ge 0$ has a density given by $\delta = \sqrt{x^2 + y^2 + z^2}$.

a) Set up a triple integral in spherical coordinates to find the mass of the hemisphere. Use Matlab to evaluate the iterated triple integral. (You should verify the result by hand.)

b) Determine the height $z = z_0$ such that half of the mass of the hemisphere lies between $z = 0$ and $z = z_0$. Use Matlab to carry out any necessary computations. Check your work by hand!

Exercise 9.4 Evaluate the triple integral $\int_0^3 \int_0^2 \int_0^1 x^2 + y^2 + z^2 dxdydz$ using **four** different methods in Matlab:

a) Exactly, using symbolic integration with the `int` command.

b) numerically, using the command `triplequad`.

c) numerically, using the command `quadvec`.

d) writing your own routine, `trpmidpoint` to estimate the mid-point Riemann sum approximation to the triple integral. Model your code after the code for `dblmidpoint`.

Include the results of using the first three methods in one M-file, clearly separating the parts and their output using cell dividers. (See note in Exercise 9.1) Submit the results using method d) in a separate M-file that contains the code for implementing the computation.

Exercise 9.5 Referring to the plot described in Exercise 7.6, using double integrals find the volume of the region inside both cylinders and lying above the *xy*-plane. Your M-file should generate

a) The picture of the 3D region whose volume is requested (see Exercise 7.6)

b) the picture of the 2D region of integration

c) the volume computed using iterated integration and the `int` function.

Your M-file should be divided into cells with the output requested above apppearing after the appropriate cell.

Exercise 9.6 Referring to the plot described in Exercise 7.8, find the volume of the region inside the sphere and above the *xy*-plane that lies within the cylinder. Your M-file should generate

a) The picture of the 3D region whose volume is requested (see Exercise 7.8)

b) the picture of the 2D region of integration

c) the volume computed using iterated integration and the `int` function.

Your M-file should be divided into cells with the output requested above apppearing after the appropriate cell.

10: Sequences and Series

Infinite series play an important role in many areas of applied mathematics, such as probability theory, differential equations, and signal processing, to name a few. In addition, the theoretical questions related to the notion of convergence provide the basis for the underlying theory of calculus, on which much of modern mathematics rests.

Matlab cannot help you determine what convergence test to use to justify the convergence or divergence of a particular sequence or series. It can however do two things that are both useful. First, it can compute accurate numerical approximations that can help you assess the existence of a limit and its numerical value. Second, using the Symbolic Toolbox, it can in many cases provide you with a symbolic expression for the value of a limit. In this chapter, we will consider how to use Matlab in each of these ways.

10.1 Numerical Estimation

10.1.1 Sequences

A sequence is an ordered list of (real) numbers, which we usually refer to in a subscript notation as $a_1, a_2, \ldots, a_n, \ldots$. From a more mathematical perspective, a sequence defines a function, *a*, whose domain is the set of natural numbers. The value of this function at $n = 1$ is written as a_1, instead of the more traditional $a(1)$, similarly a_2 instead of $a(2)$, etc. From the point of view of Matlab, we can therefore define a sequence as we do any other function, except we will use letters such as *k*, and *n*, for the independent variables, rather than *x* and *y*, and we will also often use letters such as *a* and *b* for the function names, rather than *f* and *g*.

Example 10.1: Define the sequence $a_k = \frac{(-1)^k k}{k^2+1}$ as an anonymous function of *k*. Generate the first 10 values in the sequence.

Solution: Since we want to generate a list of values, we should define the function as a vectorized expression.

```
a=@(k)(-1).^k.*k./(k.^2+1)


a =

    @(k)(-1).^k.*k./(k.^2+1)
```

Now we evaluate this function on the sequence 1:10 to find its first ten values.

```
a(1:10)


ans =

  Columns 1 through 7

   -0.5000    0.4000   -0.3000    0.2353   -0.1923    0.1622   -0.1400
```

```
Columns 8 through 10

  0.1231   -0.1098    0.0990
```

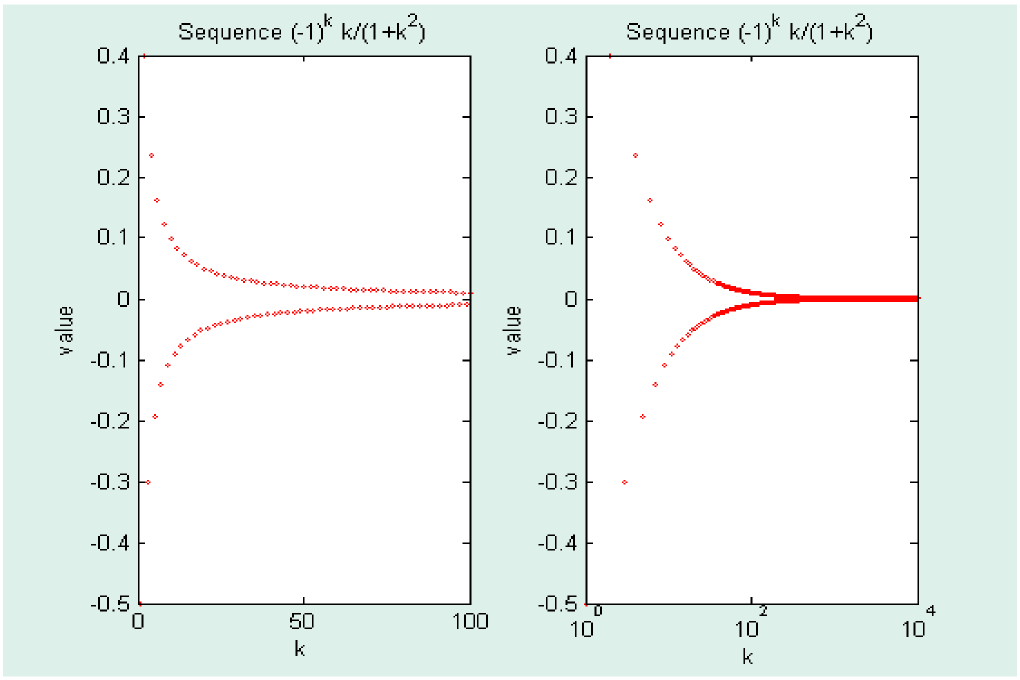
As expected, the terms alternate in sign and are getting smaller in magnitude as k increases. ■

In this example, it is mathematically pretty obvious that $\lim_{k\to\infty}\frac{(-1)^k k}{k^2+1}=0$, since the quotient $k/(1+k^2)$ tends to zero. We can get a visual sense of that behavior by plotting the values of the sequence a_k for a large run of values of k. We do this is in the following example, using the first 100 values in the sequence (not actually a very long run) and then using 10,000 values.

Example 10.2: Plot the first 100 and then the first 10,000 values of the sequence $\frac{(-1)^k k}{1+k^2}$.

Solution: We generate two plots, placing them side by side using the `subplot` command.

```
clf;
n=100;
a=@(k)(-1).^k.*k./(k.^2+1);
vals=a(1:n);
subplot(1,2,1);
plot(vals,'or','MarkerSize',2); %plot sequence values using red 'o'
marker, with no connecting line
title('Sequence (-1)^k k/(1+k^2)');
xlabel('k');ylabel('value');
n=10000;
vals=a(1:n);
subplot(1,2,2);
semilogx(vals, 'or', 'MarkerSize',2); %Log scale on k axis
title('Sequence (-1)^k k/(1+k^2)');
xlabel('k');ylabel('value');
```



Note that the plot commands did not specify the values of the independent variable. If these are not given, Matlab plots the sequence values against 1,2,…n, where n is the number of entries, which is what we wanted anyway. We did not connect the points, because technically the sequence only consists of the values generated. There is nothing wrong with connecting the points as a visual guide, as long as you realize that the segments joining successive points have no other meaning.

In the second plot we have used a semi-log plot in which the k axis is plotted on a logarithmic scale. In this type of plot, successive powers of 10 are equally spaced. Such plots (and the related `loglog` plot) are useful when the domain and/or range encompass values spread over several orders of magnitude (powers of 10). The behavior within each decile (power of ten) is then more clearly displayed. Note that between 1 and 10 we only have 10 values to plot, which show up as discrete points. In the interval from 11 to 100 there are 90 values, and many more in the larger deciles. Squeezing these into the allotted space produces the appearance of a smooth graph.

Both plots clearly show that the numbers in the sequence are approaching zero or at least some numerically very small value. ■

We have examined a sequence given by an explicit formula. An alternate way to generate a sequence is through recursion. In this process, we state values for one or more initial terms of the sequence. The recursive definition shows how to compute additional terms. For example, the famous Fibonacci sequence is determined by the recursion:

$$f_1 = 1, \quad f_2 = 1, \quad f_k = f_{k-1} + f_{k-2} \text{ if } k \geq 3.$$

From the last formula or recursion, we have $f_3 = f_2 + f_1 = 2$, $f_4 = f_3 + f_2 = 2+1 = 3$, $f_5 = f_4 + f_3 = 3+2 = 5$, and so on. If we wished to find f_{100}, we would need to find all the Fibonacci numbers for $k < 100$, which makes for considerably more work than when we examined the explict sequence in Example 10.1. Nonetheless, here are the commands of a short M-file to compute that result. The reader ought to enter this as an M-file and observe how the Matlab editor handles the looping "for…end" command when it is typed.

Example 10**.3:** Compute the 100th Fibonacci using the Symbolic Toolbox.

Solution: We use the symbolic toolbox so as to compute the exact integer value. The program computes a vector *f* whose components are the first 100 Fibonacci numbers:

```
clear f ;

% Clears f of any existing values. Useful when dealing with vector
quantities so that previously computed components are not accidentally
retained.

n=100; % the number of Fibonnaci numbers you want to compute.

f(1)=sym(1);f(2)=sym(1);

% Initialize the list to symbolic integer values using sym.  Further
calculations will automatically be done in exact arithmetic.

for k=3:n

  f(k)=f(k-1)+f(k-2);

          % recursive step; Note semicolon to suppress output.

end;      % for loop to repeatedly execute recursive statement

f(n)   % display final value


ans =

354224848179261915075
```

As you can see the number is quite large. If we had run the program by setting `f(1)=1` and `f(2)=1` the computation would have run in normal machine arithmetic and we would have obtained an approximation to the above value, namely

```
double(ans)


ans =

  3.5422e+020  .
```

■

Recursively defined sequences play an important role in many areas of mathematics, but we only mention them here for the sake of completeness. From a computational point of view they present more of a challenge than explicitly defined sequences, because at each stage the next computed value depends on previously computed values. Using the usual floating point arithmetic on the computer this may lead to magnification of small round-off errors at each stage into significant errors in the long term computation.

10.1.2 Series

We can associate with a numerical sequence a_k a new sequence, called the sequence of partial sums, defined by

$$\begin{aligned}s_1 &= a_1\\ s_2 &= a_1 + a_2\\ s_3 &= a_1 + a_2 + a_3\\ &\cdots\\ s_n &= a_1 + a_2 + \cdots + a_n\\ &\cdots\end{aligned}\quad .$$

If the sequence of these partial sums has a limit *L* we say the the infinite series $\sum_{k=1}^{\infty} a_k$ converges to *L*. We write this as $\sum_{k=1}^{\infty} a_k = L$.

The partial sum sequence can also be defined recursively by $s_1 = a_1$ and for $n \geq 2$, by $s_n = s_{n-1} + a_n$. This definition assumes that the values of a_n have been computed elsewhere (or are given by some formula that can be evaluated for each *n*).

Concretely, you can think of the relationship between the sequences a_k and s_k as follows. The numbers a_k represent weekly deposits (if positive) or withdrawals (if negative) from a bank account. The partial sums give the total accumulation in the account each week. How does your pattern of weekly deposits affect the pattern of your total accumulations? Well, the theory of infinite series you are studying in class tries to provide some answers to that question. Here we will use Matlab to investigate the sequence of partial sums in a manner similar to our earlier investigation of sequences.

The principal tool in doing this is the command `cumsum`, short for cumlative sum. Following our analogy with a bank account, this simply computes the accumulated sum in your account at the end of each time period. For example, with

```
x=[1 2 3 4];
```

we have

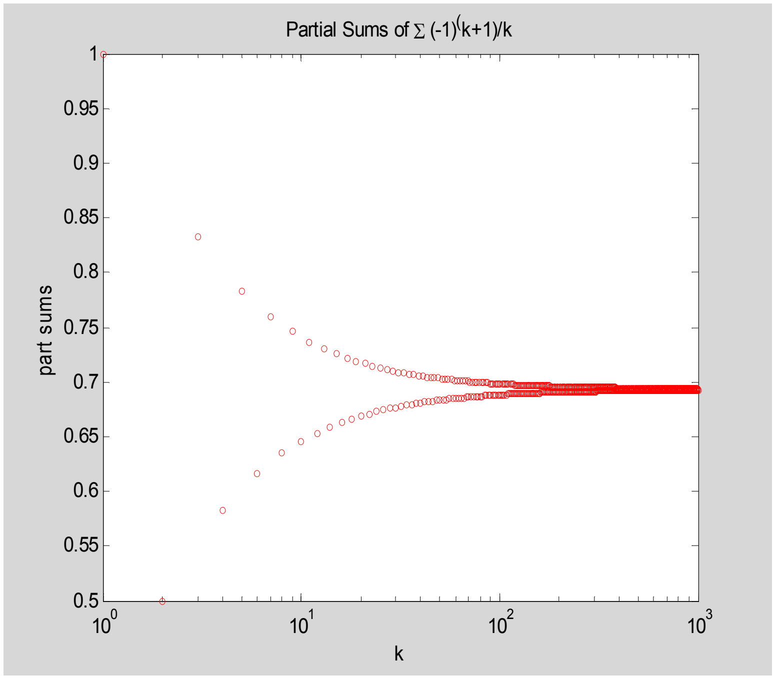
```
cumsum(x)


ans =

     1     3     6    10 .
```

In short, `cumsum` generates the sequence of partial sums associated with the infinite series $\sum_{k=1}^{\infty} a_k$. We can plot this sequence to generate a visual picture of the behavior of the partial sum sequence.

Example 10.4**:** Make a semilog plot of the partial sums of the first 1000 terms of the series $\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} = 1 - \frac{1}{2} + \frac{1}{3} + \cdots$. Does the plot give evidence of the convergence of the series?

Solution:

```
clf
a=@(k)(-1).^(k+1)./k; % define the sequence
n=1000;
vals=a(1:n);
s=cumsum(vals);
semilogx(s, 'or', 'MarkerSize',2)
xlabel('k');ylabel('part sums');
title('Partial Sums of \Sigma (-1)^(k+1)/k')
% \Sigma codes for Greek letter upper case sigma
```



The series seems to be approaching a limit. The term

```
s(1000)
```

is

```
ans =
    0.6926
```

and based on the graph, this should be a good approximation to the limit of the series, though this cannot be ascertained from the numerical computation alone. Comparing this partial sum to the previous one

```
s(999)


ans =

    0.6936
```
,

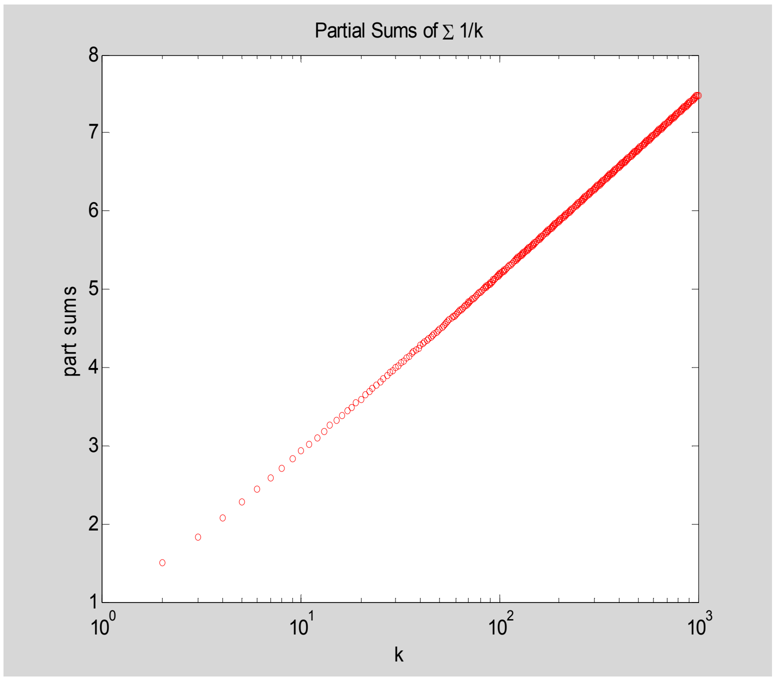
we have some confidence in asserting that the limit value is between .692 and .694, or in less precise terms $.693 \pm .001$. In this case, the error estimates associated with alternating series provide a rigorous justification for this conclusion. ■

The harmonic series $\sum_{k=1}^{\infty} 1/k$ is an important example of a divergent series whose terms go to zero. Is such behavior detectable using Matlab?

Example 10.5: Make a semilog plot of the partial sums of the first 1000 terms of the harmonic series. Discuss the convergence of the series as it relates to the plot.

Solution:

```
clf
a=@(k)1./k; % define the sequence
n=1000;
vals=a(1:n);
s=cumsum(vals);
semilogx(s, 'or', 'MarkerSize',2)
xlabel('k');ylabel('part sums');
title('Partial Sums of \Sigma 1/k')
```



The graph is strikingly different from the graphs for the partial sums of a convergent series. While the picture does not preclude that the partial sums converge, it certainly provides no compelling reason to think so, and a strong suggestion that they do not. Nonetheless, only a mathematical argument based on examining properties of the sequence can actually settle that question. ■

10.2 Symbolic Limits

The Symbolic Toolbox in Matlab is capable of finding limits of many sequences and series. These may be expressed in terms of familiar functions and mathematical constants, or may involve values of special functions. Many of these results rest on mathematical analysis that is quite advanced and well beyond the level of this course. However, these results are of interest because they often point to mysterious connections between seemingly unconnected parts of mathematics. The exercises will present more examples for the reader to consider.

Example 10.6: Find the value of $\lim_{k\to\infty}\left(1+\frac{1}{k}\right)^k$.

Solution:

First declare *k* symbolic. Then use the `limit` function.

```
syms k
limit((1+1/k)^k,inf) % inf is the Matlab abbreviation for "infinity"
```

```
ans =
exp(1)
```

The answer, which is *e*, is expressed as a value of the exponential function. ■

More impressive are the results obtained for infinite series. The command `symsum` computes the symbolic limit.

Example 10.7: Find the value of $\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k}$.

Solution:

```
syms k
symsum((-1)^(k+1)/k,1,inf) % last two arguments give the summation range
```

```
ans =
log(2)
```

The answer is expressed in term of the natural log function. Note the value is

```
double(ans)
```

```
ans =
    0.6931
```

The reader should compare this to the numerical approximation computed in Example 10.4. ■

10.3 Summary

A sequence is a function whose domain is the set of natural numbers. Such functions can be defined using the standard method of anonymous functions or can be constructed through recursive definitions.

We can analyze the limit behavior of sequences and series through numerical computation. The results can be usefully presented in graphical format. The Symbolic Toolbox provides commands that can be used in many cases to find exact formulas for limits.

The following commands were introduced in this chapter:

`semilogx`: - plot with independent variable displayed in a logarithmic scale

`cumsum`: - computes sequence of partial sums of a numerical sequence

`limit`:- compute a symbolic limit of an explicitly given sequence

`symsum`: - compute a symbolic expression for a sum of terms of an explicitly given sequence

10.4 Exercises

Exercise 10.1 Let $a_k = k^{1/k}$. Write an M-file that

a) plots (you can use a regular plot, instead of a semilog plot) the first 100 values of the sequence as individual points

b) based on the numeric values generated makes an estimate of the limit of the sequence

c) uses the Symbolic Toolbox to find the exact limit.

Publish your M-file. The results of a), b), and c) should appear as separate output from different cells in the file. Don't forget to include your name at the top.

Exercise 10.2 Repeat the instructions in Exercise 10.1 for the sequence $a_k = \left(1 - \frac{2}{k}\right)^k$.

Exercise 10.3 Investigate which if any of the following sequences have limits. Provide all possible justifications for your answer using a variety of Matlab computations and graphs. Present your work in the form of a published M-file with appropriate comments:

a) $a_k = \sin(k)$

b) $b_k = (\sin(k))^k$

c) $c_k = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{k} - \ln k$

Exercise 10.4 Every year your bank account earns 5% interest on the principal on deposit for that entire year. In addition, you make an additional deposit of \$1000 into the account at the end of each year and no withdrawals. Let P_n denote the principal in the account at the beginning of year *n*.

a) For $n \geq 2$, express the principal P_n in terms of the principal P_{n-1}.

b) Assuming your initial principal is $P_1 = \$1000$, find the principal at the beginning of year 21. How much of that principal is interest? Make a graph showing the growth of the principal from year 1 to year 21.

c) Assuming the same initial deposit of \$1000 and a 5% interest rate, what is the minimum you would have to deposit each year (assuming equal deposits) in order for your accumulation to reach \$50,000 by the beginning of year 21? Display your result graphically.

Exercise 10.5 Write an M-file to generate a `semilogy` plot of the first 100 Fibonacci numbers. What can you deduce from the pattern of the plot for large *k*? Extend the plot for a larger range of the index. Use your plot to develop an approximate explicit formula for the *k*th Fibonacci number.

Exercise 10.6 Modify the Fibonacci sequence so that for $n \geq 3$ the recursion is $f_k = f_{k-1} + f_{k-2} + (-1)^k k$. How large is the 100th value of this sequence? Produce an appropriate plot showing the values of the numbers in this sequence.

Exercise 10.7 Investigate the convergence or divergence of each of the following series. You should use numerical and/or symbolic computations to arrive at your conclusions. To the extent possible you should confirm your answers using the theory covered in the course. Present the response to each part as a separate published M-file.

a) $\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2^k}$

b) $\sum_{k=1}^{\infty} \frac{1}{k!}$ (The quantity $k!$ can be entered in Matlab as `factorial(k)`.)

c) $\sum_{k=1}^{\infty} \frac{\sin(k)}{k}$

d) $\sum_{k=1}^{\infty} \frac{\sin(k)}{k^2}$

e) $\sum_{k=1}^{\infty} \frac{1}{k(k+1)(k+2)}$

f) $\sum_{k=1}^{\infty} \frac{1}{k^2}$

g) $\sum_{k=1}^{\infty} \frac{(-1)^k}{k^2+1}$

11: Power Series

One of the main applications of the theory of numerical series is to the study of series of functions. In particular, when the functions take the simple form of powers of the variable multiplied by a constant, we speak of a power series. Such series are a workhorse of many numerical methods in mathematics because they allow one to approximate complicated functions by much simpler ones. Infinite series comprised of trigonometric functions, so-called Fourier series, play a similar role when studying periodic phenomena.

Matlab is quite useful in helping us visualize the convergence process and approximating a limit when one exists. In many cases, it can also provide closed formulas for the limit function of an infinite series. We will use the Symbolic Toolbox in our analysis, since restricting the treatment to just numerical manipulations in Matlab presents some technical difficulties that would obscure the mathematical ideas.

11.1 Plotting Power Series

A power series is an infinite series of the form

$$\sum_{k=0}^{\infty} a_k (x-c)^k$$

where the a_k and c are real numbers. The principal question concerning such an object is determining the set of values of the variable x for which the series converges. The theory of series informs us that convergence occurs in some interval of length $2R$ centered around the point c, where R may be zero or infinity. This is called the *interval of convergence* and the quantity R is called *the radius of convergence*. If $0 < R < \infty$, further analysis is required to determine whether the series converges at the endpoints $c-R$ and $c+R$.

Assuming you have determined the interval of convergence for a particular power series, we will write an M-file that plots the series and a partial sum $\sum_{k=0}^{n} a_k (x-c)^k$, where n is chosen by the user. We will consider the example

$$\sum_{k=1}^{\infty} \frac{2^k}{k} x^k ,$$

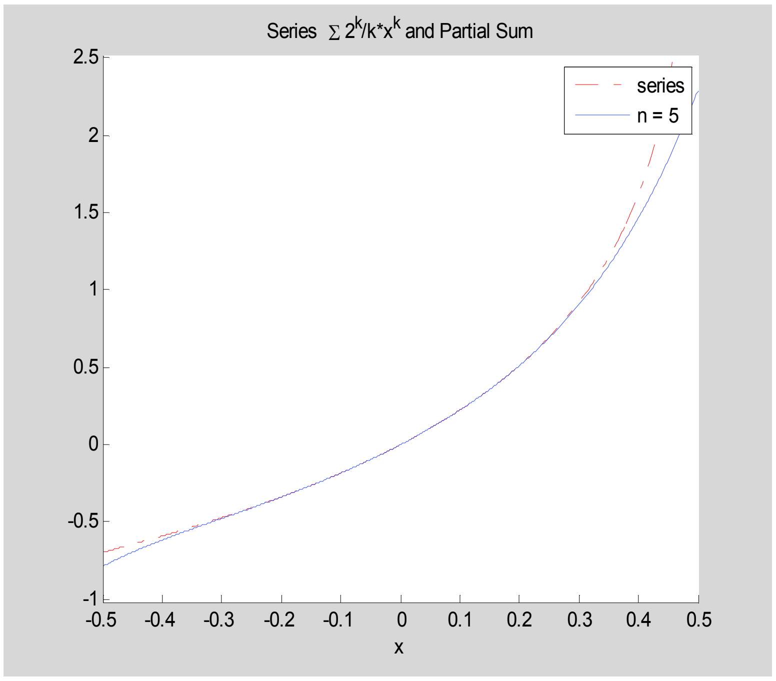
which converges in the interval $|x| < \tfrac{1}{2}$.

Example 11.1: Plot the limit of the series $\sum_{k=1}^{\infty} \frac{2^k}{k} x^k$ and the partial sum $\sum_{k=1}^{5} \frac{2^k}{k} x^k$ on the interval $[-1/2, 1/2]$.

Solution:

Since we will be generating and plotting symbolic expressions we will be working with `ezplot`, which we studied at the beginning of the course.

```
syms k x
clf; hold on
%%%%% User Input %%%%%%%
a=@(k)2^k/k;  % define coefficient sequence
f=@(x)symsum(a(k)*x^k,k,1,inf); %define series
p=@(x,n)symsum(a(k)*x^k,k,1,n); % define partial sum
n=5;  % degree of partial sum
x1=-1/2;x2=1/2; % endpoints of plotting interval
%%%%% End User Input %%%%%%%
h=ezplot(f(x), [x1,x2]);  % plot limit function
set(h, 'Color','r', 'LineStyle','-.'); % use handle to set properties
ezplot(p(x,n), [x1,x2]); % plot partial sum; leave default style
legend('series',['n = ', num2str(n)]);
% num2str converts n into a string
title(['Series  \Sigma ',char(a(k)), '*x^k and Partial Sum' ]);
hold off
```



The two graphs are almost indistinguishable throughout the interval of convergence. The `symsum` command actually computes the limit of the power series. Namely,

```
symsum(2^k/k*x^k, k, 1, inf)


ans =

-log(1-2*x)
```

For convenience, this M-file is available in the course M-file folder as `powseries.m`. Note, this M-file will only work for power series for which Matlab is able to find a closed form expression. For example, even though the series $\sum_{k=1}^{\infty}\frac{x^k}{\sqrt{k}}$ converges on the interval $(-1,1)$, Matlab cannot determine an explicit formula for the series and therefore is unable to execute the `ezplot` command. ■

11.2 Taylor Series

A power series determines a function on its interval of convergence. We can recover the power series coefficients from this function by successive differentiation. Namely, if

$$f(x)=\sum_{k=0}^{\infty}a_k(x-c)^k$$

then $a_k=\frac{f^{(k)}(c)}{k!}$. If we only have the function $f(x)$ to begin with, we can use the latter formula for a_k (provided *f* is infinitely differentiable) to define a power series. This series is called the *Taylor series* of the function *f* around $x=c$, or in powers of $x-c$. For most functions this series converges to the original function, at least on the interior of the interval of convergence for the series. The partial sum,

$$p_n(x)=\sum_{k=0}^{n}\frac{f^{(k)}(c)}{k!}(x-c)^k$$

is called a *Taylor polynomial* (of degree *n*).

We illustrate computations with Taylor polynomials in Matlab and then leave it to the reader to exhibit graphically the convergence of these series through a simple modification of our previous program.

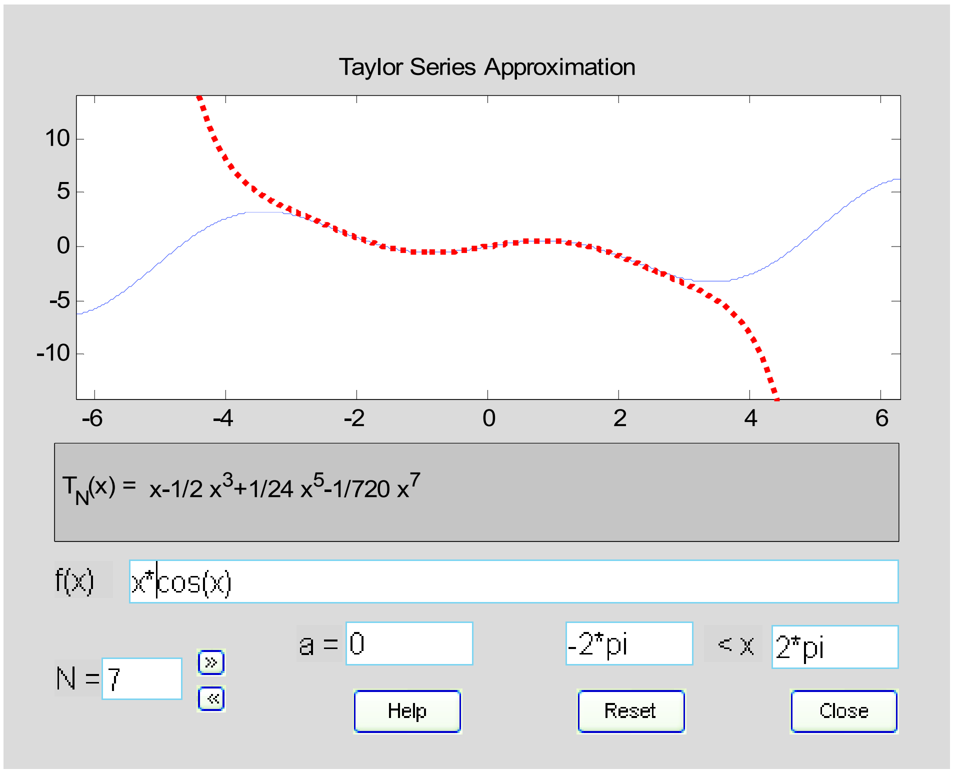
Example 11.2: Let $f(x)=\sin x$. Find the Taylor polynomial $p_9(x)$ with center $c=0$ (i.e. Maclaurin series) and compare the values of $p_9(\pi/2)$ and $\sin(\pi/2)=1$.

Solution: The `taylor` command computes Taylor polynomials for a given function. We convert the output to a polynomial function using the method described in Example 6.6.

```
syms x
tpoly=taylor(sin(x),10)  % defines taylor polynomial expression
p=@(x)subs(tpoly);  % taylor polynomial function
approx_error= abs(p(pi)-sin(pi)) % absolute difference in approximation

tpoly =
x-1/6*x^3+1/120*x^5-1/5040*x^7+1/362880*x^9
approx_error =
    0.0069
```

■

Although it is a straightforward matter to modify the M-file `powseries.m` to have it plot a function and its Taylor polynomial, we can save ourselves some work by using a built-in Matlab M-file for that purpose. This is the file `taylortool`. When entered from the command line, this brings up an interactive dialog box as shown below:



$T_N(x) = x-1/2\,x^3+1/24\,x^5-1/720\,x^7$

f(x) x*cos(x)

N = 7

a = 0

-2*pi < x 2*pi

The user can change the function and various other settings and observe the effect on convergence.

11.3 Summary

Power series can be easily defined through the Symbolic Toolbox using the `symsum` command. The `taylor` command generates Taylor polynomials.

The following commands were introduced in this chapter:

`taylor`: - generate taylor polynomial for a symbolic expression. See help page for options to specify center and degree of expansion.

`num2str`: - convert a number to a string. This is useful in constructing informative titles and legends that contain a reference to a numerical parameter. Usually this is concatenated with other text using [text1, text2, …] to produce a meaningful annotation.

11.4 Exercises

Exercise 11.1 Find the interval of convergence for each of the following series. (Use the usual methods to do so.) Modify the file `powseries.m` so as to generate two plots: the first plot showing the series and an appropriate partial sum, and the second, in a separate window, showing the absolute difference (error) between the two functions.

Select your partial sum so it provides a reasonable approximation to the series throughout most of the interval of convergence. Publish your result. Make sure your graphs have clear titles that explain their content.

a) $\sum_{k=1}^{\infty} (-1)^k \frac{x^k}{k^2}$

b) $\sum_{k=0}^{\infty} \frac{3^k x^k}{2k+1}$

Exercise 11.2 The file `powseries.m` only works for power series for which Matlab can find an explicit sum. Modify the file so that instead of drawing the graph of the entire power series and the *n*th partial sum, it draws the graphs of the *n*th and (2*n*)th partial sums over the convergence interval. Don't forget to modify the legend and title appropriately.

In each of the following examples, determine the interval of convergence of the series in the usual way. Then use your modified M-file to plot a pair of partial sums that exhibit the convergence of the partial sums to a limit.

a) $\sum_{k=1}^{\infty} (-1)^k \frac{x^k}{\sqrt{k}}$

b) $\sum_{k=0}^{\infty} \frac{x^k}{4+k^2}$

Exercise 11.3 Modify the M-file `powseries.m` so that it displays the graph of the function $f(x) = x \sin x$ over the interval $[-2\pi, 2\pi]$ and the graph of the Taylor polynomial with center $c = 0$ and degree 8. Use an appropriate title and legend. Organize the file so that the data on the function, plotting interval, center of the expansion, degree of the approximation appear in a clearly marked "User Entry Section" near the beginning. Publish your file and its output to html.

Exercise 11.4 The Lagrange error estimate says that

$$\left|f(x)-p_n(x)\right| \le \frac{\left|f^{(n+1)}(\theta)\right|}{(n+1)!}\left|x-c\right|^{n+1} \le \frac{M_{n+1}}{(n+1)!}\left|x-c\right|^{n+1}$$

where, $p_n(x)$ is the degree *n* Taylor polynomial at with center *c*, θ is some number between *x* and *c*, and M_{n+1} is the maximum absolute value of the $n+1$st derivative on the interval containing *x* and *c*. We can use Matlab to estimate this error estimate and compare it to the actual error.

In each of the following, make a plot of the appropriate derivative function to estimate its maximum absolute value on the stated interval. From this result determine an upper bound for the Lagrange error estimate and compare this upper bound with the absolute difference between $f(x)$ and $p_n(x)$ found by plotting $\left|f(x)-p_n(x)\right|$. Publish each example in a separate M-file.

a) $f(x)=\sin x$, $c=0$, $n=3$, interval: $[-.5,.5]$

b) $f(x)=\tan x$, $c=0$, $n=5$, interval: $[-.5,.5]$

c) $f(x)=\sin x$, $c=\pi/4$, $n=5$, interval: $[40^\circ, 50^\circ]$ (N.B. You will need to convert to radians)

d) $f(x)=\ln(1+x)$, $c=0$, $n=7$, interval: $[0,.5]$