Maple Lecture 16. Maple Procedures and Recursion

Maple procedures can take procedures as input and give procedures on return. We will also see how to work with indexed procedures. With a remember table we can make recursive procedures to run efficiently.

The material in this lecture is inspired on [2, Section 8.4]. The first example below is taken from [3, pages 75-77], see also [1, Section 3.5] for recursion and remember tables. The most recent information can be found in the Maple 9 manuals [4] and [5]. The usage of remember tables is also known as memoization.

16.1 Procedures returning Procedures

Newton's method is one of the most fundamental algorithms for approximating solutions of \( f(x) = 0 \), where the approximations are generated as follows:

\[
  x(k+1) = x(k) - \frac{f(x(k))}{f'(x(k))}, \quad \text{for } k = 0, 1, \ldots
\]

where \( f'(x) \) is the derivative of the function \( f \).

We will make a procedure that returns the right hand side of the iteration above. First of all, we must note the difference between \( x \) and \( x \rightarrow x \): the first \( x \) is just the name \( x \), while \( x \rightarrow x \) is the function \( x \).

\[
\begin{aligned}
  \text{> newtonstep := proc(f::procedure) \\
  \text{\quad description 'returns one step with Newton's method on f':} \\
  \text{\quad local ix:} \\
  \text{\quad ix := x -> x: \# identity function} \\
  \text{\quad ix - eval(f)/D(eval(f)); \# implicit return} \\
  \text{\quad end proc;} \\
\end{aligned}
\]

Note that we use the \text{eval} in the procedure to force Maple to evaluate, because for efficiency, Maple would otherwise delay the evaluation. Let us apply this to approximate a root of \( \cos(x) = 1/2 \). First we must make a function \( g(x) = \cos(x) - 1/2 \).

\[
\begin{aligned}
  \text{> g := x -> cos(x) - 1/2; \# compute root of } g(x) = 0 \\
  \text{> gstep := newtonstep(g); \# create a procedure} \\
  \text{> gstep(a); \# symbolic execution} \\
  \text{> gstep(1.4); \# numerical execution} \\
  \text{> y := 0.4; \# starting value} \\
  \text{> Digits := 32; \# working precision} \\
  \text{> for i from 1 to 7 do} \\
  \text{\quad y := gstep(y); \# we will do 7 steps} \\
  \text{> end do;} \\
\end{aligned}
\]

We know that \( \cos(\pi/3) = 1/2 \), let us thus check how accurate our result is:

\[
\text{> evalf(y - Pi/3);}
\]

16.2 Indexed Procedures

An example of an indexed procedure is the logarithm, where the base can be given as an index.

\[
\begin{aligned}
  \text{> interface(verboseproc=3);} \\
  \text{> print(log);} \\
\end{aligned}
\]

By default, we get the natural logarithm:

\[
\begin{aligned}
  \text{> log(10.0); log(exp(1));}
\end{aligned}
\]
To get the decimal logarithm, we need to provide the base 10 of the logarithm as index to the function call:

\[ \log_{10}(10.0); \]

An index is just like an index in an array:

\[ a := A[3]; \]
\[ \text{type}(a, '\text{indexed}'); \]
\[ \text{op}(a); \]

We see that we can check on whether a name is indexed or not via type and get access to the index with op.

As example, suppose \( f(t) = b + (70 - b) \cdot \text{exp}(-0.2 \cdot t) \) models temperature in function of time with \( b \) as index. Initially, at \( t = 0 \), the temperature is 70. As \( t \) goes to infinity, the final temperature is \( b \). If \( b \) is not provided as index, take \( b = 32 \) as default.

\[ \text{cool} := \text{proc}(t) \]
\[ \quad \text{description} \ '\text{model of cooling temperature with index}'; \]
\[ \quad \text{local} \ b; \]
\[ \quad \text{if type(procname, 'indexed')} \# \text{test if procedure has an index} \]
\[ \quad \quad \text{then} \ b := \text{op(procname)}; \# \text{take index as base} \]
\[ \quad \quad \text{else} \ b := 32; \# \text{default value of base} \]
\[ \quad \quad \text{end if}; \]
\[ \quad \quad \text{return} \ b + (70-b) \cdot \text{exp}(-0.2 \cdot t); \# \text{the general formula} \]
\[ \quad \end \text{proc}; \]

\[ \text{cool}[20](1.4); \text{cool}(1.4); \# \text{test for different values of base} \]
\[ \text{cool}[20](0); \text{cool}(0); \# \text{initially we are inside} \]
\[ \text{cool}[20](100); \text{cool}(100); \# \text{close to outside temperature} \]

We use indexed procedures to implement functions with parameters for which good default values are known. The default values may correspond to cases for which a very efficient implementation exists, whereas for other values, a general recipe needs to be applied.

16.3 Recursive Procedure Definitions

Many functions are defined recursively. We see how Maple has a nice mechanism to avoid superfluous recursive calls. One classical example of a recursive sequence are the Fibonacci numbers:

\[ F(0) = 0, \quad F(1) = 1, \quad \text{and} \quad F(n) = F(n-2) + F(n-1), \quad \text{for} \ n \geq 2. \]

The direct way to implement this goes as follows:

\[ \text{fib} := \text{proc}(n::\text{nonnegint}) \]
\[ \quad \text{description} \ '\text{returns the nth Fibonacci number}'; \]
\[ \quad \text{if} \ n = 0 \ \text{then} \]
\[ \quad \quad \text{return} \ 0; \]
\[ \quad \quad \text{elif} \ n = 1 \ \text{then} \]
\[ \quad \quad \quad \text{return} \ 1; \]
\[ \quad \quad \text{else} \]
\[ \quad \quad \quad \text{return} \ \text{fib}(n-2)+\text{fib}(n-1); \]
\[ \quad \quad \end \text{if}; \]
\[ \quad \end \text{proc}; \]

\[ \text{for} \ i \ \text{from} \ 1 \ \text{to} \ 10 \ \text{do} \quad \# \text{first ten Fibonacci numbers} \]
\[ \quad \text{fib}(i); \]
\[ \end \text{do}; \]

This is a very expensive way to compute the Fibonacci numbers, because of too many repetitive calls.
In Figure 1 we see the tree of procedure calls to compute $F(4)$. In general, to compute the $n$th Fibonacci number, $2^n$ calls are needed.

![Figure 1: Procedure Calls to compute $F(4)$.](image)

We will slightly modify the definition of the procedure to compute the Fibonacci numbers:

```maple
> newfib := proc(n::nonnegint)
> description 'Fibonacci with remember table':
> option remember:
> if n = 0 then
> return 0;
> elif n = 1 then
> return 1;
> else
> return newfib(n-2) + newfib(n-1);
> end if:
> end proc;
> starttime := time():
> newfib(20);
> elapsed := (time()-starttime)*seconds;
```

With the option remember, Maple has built a “remember table” for the procedure. This remember table stores the results of all calls of the procedure. Here is how we can consult this table:

```maple
> eval(newfib);
> T := op(4,eval(newfib));
```

If you are curious about the “4”, do `?proc;` to see where the other operands are used for. With calls to `newfib` for higher numbers, we add values to the table:

```maple
> newfib(21);
> eval(T);
```

Once we selected the remember table and assigned it to a variable, we can modify the table.

```maple
> newfib(20) := 1; # introduce error in the table
> eval(T);
```

We can also unassign values in the table:

```maple
> unassign(T);
```
As the computation of the 22nd Fibonacci number required the 20th, the 20th element has been recomputed and stored in the remember table:

\[
\text{eval(T;}
\]

The command `forget` is used to clear the remember table of a Maple procedure. For example:

\[
\text{forget(newfib;}
\]

**16.4 Assignments**

1. Write a procedure `fractinal_power` which returns \(x^{1/n}\) for one argument \(x\) and index \(n\). If the index is omitted, `fractinal_power(x) = \sqrt{x}`.

2. Indices can be sequences. Write a procedure `line` which has one argument \(x\) and up to two indices. The output of `line` is as follows: `line[a, b](x) = a + bx`, `line[a](x) = a_1 + a_2x`, and `line(x) = x`.

3. The secant method to find a solution of \(f(x) = 0\) is defined by

\[
x_n = x_{n-1} - \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} f(x_{n-1}), \quad \text{for } n \geq 2.
\]

While the secant method requires no derivatives, we need two points \((x_0, x_1)\) to start the iteration. For simplicity we will take for \(x_0\) and \(x_1\) a random float generated by `evalf(rand()/10^12`.

(a) Write a Maple procedure to implement the formula above, to execute one step of the secant method. Use the following prototype:

\[
\text{secantstep := proc(f::procedure,x0::float,x1::float);}
\]

Test your implementation on \(f(x) = \cos(x) - 1/2 = 0\).

(b) Use `secantstep` to define the Maple procedure with prototype

\[
\text{secant1 := proc(f::procedure,n::nonnegint);}
\]

which returns \(x_n\), starting from random values for \(x_0\) and \(x_1\).

Also here, test your implementation on \(f(x) = \cos(x) - 1/2 = 0\).

(c) Write a recursive implementation for the secant method, using the prototype

\[
\text{secant2 := proc(f::procedure,n::nonnegint);}
\]

which also returns \(x_n\), starting from random values for \(x_0\) and \(x_1\).

Make sure this recursive implementation is as efficient as the iterative version.

4. Execute `diff(sin(x),x);` and change the remember table of `diff` so that next time we execute `diff(sin(x),x);` we get \(\sin(x)\) on return.

5. The Bell numbers \(B(n)\) are defined by \(B(0) = 1\) and \(B(n) = \sum_{i=0}^{n-1} \binom{n-1}{i} B(i), \quad \text{for } n > 0.\) They count the number of partitions of a set of \(n\) elements.

Write a recursive procedure to compute the Bell numbers. The binomial coefficient \(\binom{n-1}{i}\) is computed by `binomial(n-1,i)`. Make sure your procedure is efficient enough to compute \(B(50)\).
6. The Stirling numbers of the first kind $c(n, k)$ satisfy the recurrence

$$c(n, k) = -(n - 1)c(n - 1, k) + c(n - 1, k - 1), \quad \text{for} \quad n \geq 1 \quad \text{and} \quad k \geq 1,$$

with the initial conditions that $c(n, k) = 0$ if $n \leq 0$ or $k \leq 0$, except $c(0, 0) = 1$.

(a) Write an \textit{efficient recursive} procedure, call it \texttt{stirling1} to compute $c(n, k)$.

The $n$ must be an index to \texttt{stirling1} while $k$ is its argument, e.g.: for $n = 100$ and $k = 33$, \texttt{stirling1[100](33)} should return $c(100, 33)$.

(b) How many digits does the number $c(100, 33)$ have? Give also the Maple command(s) to obtain this number.

7. The $n$-th Chebychev polynomial is also often defined as $\cos(n \arccos(x))$.

Give the definition of the procedure \texttt{C} which takes on input $x$ and has index $n$.

Thus \texttt{C[n](x)} returns $\cos(n \arccos(x))$ while \texttt{C[10](0.5)} returns the value of the 10-th Chebychev polynomial at 0.5. Compare this value with \texttt{orthopoly[T](10,0.5)}.

8. Let $L[n](x)$ denote a special kind of the Laguerre polynomial of degree $n$ in the variable $x$.

We define $L[n](x)$ by $L[0](x) = 1$, $L[1](x) = x$, and for any degree $n > 1$:

$$n*L[n](x) = (2*n-1-x)*L[n-1](x) - (n-1)*L[n-2](x).$$

Write a Maple procedure \texttt{Laguerre} that returns $L[n](x)$.

Use an index for the degree $n$ and take $x$ as parameter in the procedure.

Make sure your procedure can compute the 50-th Laguerre polynomial.

9. Denote the composite Trapezoidal rule for $\int_a^b f(x) \, dx$ using $2^n$ intervals by $T[n](f,a,b)$.

We can define $T[n](f,a,b)$ recursively by two rules:

$$T[0](f,a,b) = (f(a) + f(b))*(b-a)/2;$$

$$T[n](f,a,b) = T[n-1](f,a,(a+b)/2) + T[n-1](f,(a+b)/2,b), \quad \text{if} \quad n > 0.$$

(a) Write a recursive Maple procedure for $T$, where $n$ must be an index to $T$.

(b) Explain what the user of $T$ must do to prevent that $f$ is never evaluated twice at the same point.

Illustrate using $n = 5$ in $T$ for the numerical approximation of $\int_0^1 \cos(x) \, dx$.

\textbf{References}


