Homework 04 - Numerical Integration

Arturo Popoli

January 24, 2026

Visualize gauss points

Extend the implementation to return the Gauss node positions (`xg`) employed during the recursive quadrature

```
[int,xg] = int_adt(f,a,b,3,1E-6)
```

Hint: within `int_gauss(f,a,b,n)` you can get the positions of the gauss nodes in the original interval [a,b] exploiting the same linear transformation $x = m\tau + q$ that allowed us to transform [a,b] into the reference interval [-1,1]

Try this new feature solving the following integral:

$$\int_0^{2.5} e^x \, dx$$

You should obtain something similar to the following (`tol=1E-6`):

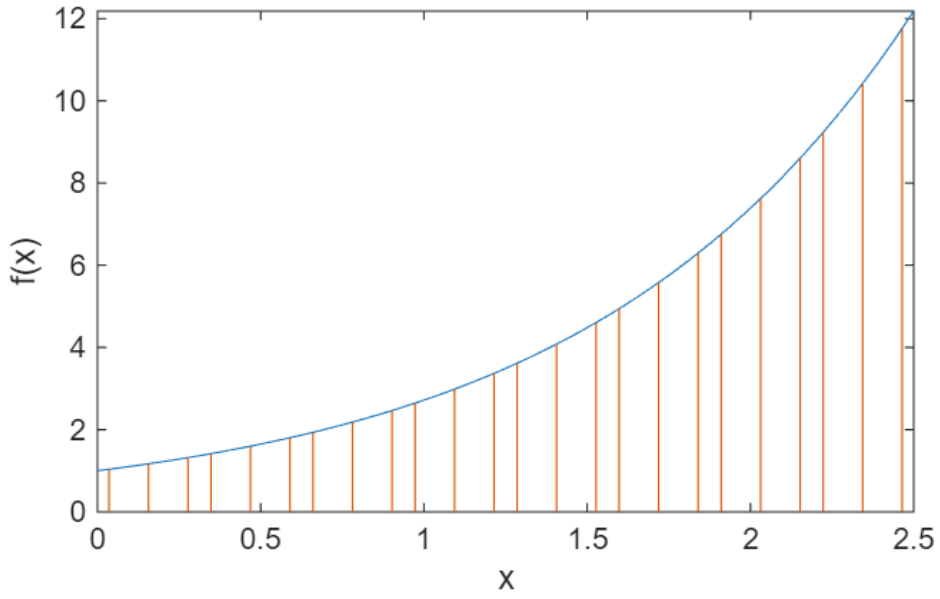

Figure 1: Example of gauss points visualization

. . . what if you now try with a more challenging function?

```
f = @(x) (x + 1).^2 .* cos((2 * x + 1) ./ (x - 4.3)); % integrand function
a = 0; b = 4;
```

As expected we get more gauss points in the regions where the function changes are more drastic!

. . . try to plot the same graph again lowering the tolerance to 1E-3 and observe the difference.

Stopping criterion

What if we consider an integral that would require too many integration points (and therefore too much computation time)?

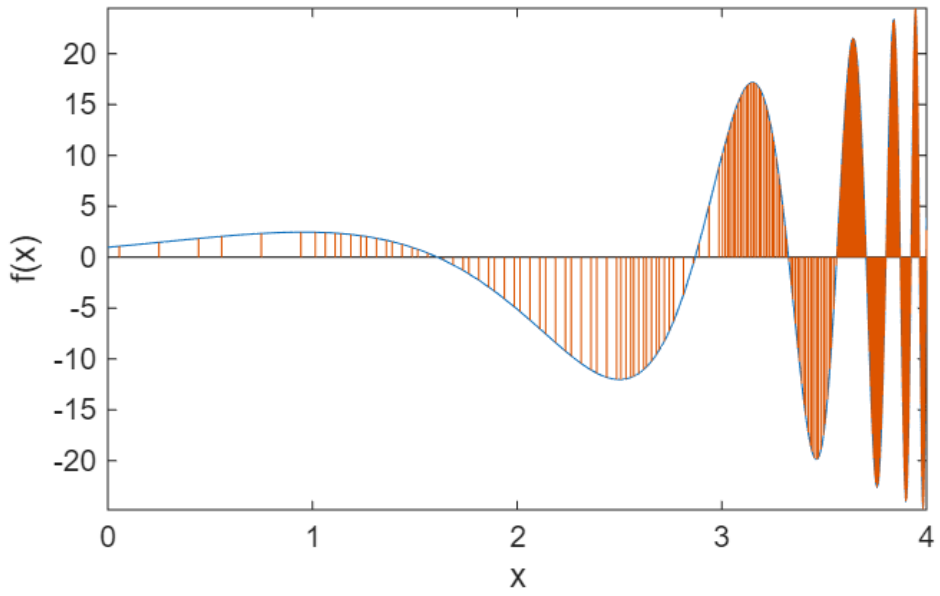

Figure 2: Example of gauss points visualization (2)

Consider, for example:

$$\int_0^{100} \sin(10^3 x)\, dx = \frac{1-\cos(10^5)}{1000} \approx 0.0019993608074382126$$

If you try to integrate that with Matlab's default you will get a warning message issued by the function itself, suggesting that a large number of function evaluations is required.

Now try to repeat the calculation with our `int_adt` function. Check how many gauss points were used and measure the execution time with the `timeit` function.

...so now extend the `int_adt` function to require an additional `nrmax` parameter defining the maximum number of recursions (which you can internally compute based on the number of entries in `xg` and `n`). We can also do that without changing the main interface for the user since one m-file can contain multiple functions.

```
function [int,xg,nr] = int_adt_nrmax(f, a, b, n, tol, nrmax)
    nr = 0;
    % Call the internal recursive worker with nr=0
    [int,xg,nr] = int_adt_w(f, a, b, n, tol, nr, nrmax)
    if nr == nrmax
      warning('nrmax reached');
    end
end

% Recursive Worker Function
function [int,xg,nr] = int_adt_w(f, a, b, n, tol, nr, nrmax)

  % your job!...

end
```

If everything works you should get something like:

```
[int,xg,nr] = int_adt_nrmax(f,a,b,3,1E-6,20);
Warning: nrmax reached
> In int_adt_nrmax (line 8)
```

Introducing argument parsing

Argument parsing is the way a function handles and validates its input parameters. In modern MATLAB, this is done with an arguments block, but the concept exists in all languages (in Python, argument parsing is done with default function arguments and optional keyword arguments).

With argument parsing you can:

- Set default values for optional inputs.
- Validate inputs automatically (type, size, range).
- Make your functions more flexible and easier to call.

For example, instead of forcing the user to always specify `n`, `tol`, and `nrmax`, you can provide sensible defaults.

Update the function `int_adt` so that it can be called with just three arguments:

```
int = int_adt(f, a, b)
```

Use the following default values:

```
n = 3
tol = 1e-10
nrmax = 1000
```

To do that, explore arguments MATLAB functionality

[OPTIONAL] further adventures for true adventurers

Let's stress-test our implementation with a very unpleseant function

```
f = @(x) exp(-(x-0.5).^2/(2*c^2))
c = 5E-4;

a = 0; b = 1;
```

This gaussian function centered in 0.5 is particularly challenging because it's very narrow, i.e., the function will be zero for most (if not all!) our gauss points.

. . . but yet Matlab's builtin integral function has no problems getting the right result,i.e.:

```
integral(f,a,b)
ans =
   0.001253314137315
```

We could try to improve the behaviour of our function by adding an optional input with a "hot" point where the user expects the function to spike (e.g., 0.5 in the example function). The code could then benefit from this information by adding some constraints to the intervals such that the routine is forced to place some gauss points around the hot point.