CS319: Scientific Computing

Week 8: Quadrature in 2D; Intro to Classes

Projects!

Dr Niall Madden

9am and **4pm**, 28 February, 2024

Slides and examples: https://www.niallmadden.ie/2324-CS319

Outline

Slides and examples:
https://www.niallmadden.ie/2324-CS319

Projects!

https://www.niallmadden.ie/2324-CS319/#projects

Quadrature 2D

(These slides we part of Week 7, but I didn't get to them in class).

For the last time (in lectures) we'll look at **numerical integration**, this time of two dimensional functions.

That is, our goal is to estimate

$$\int_{a_1}^{b_1} \int_{a_2}^{b_2} f(x_1, x_2) dx_1 dx_2.$$

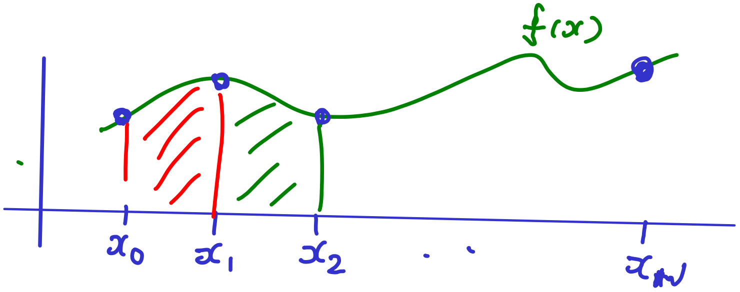
When we implement an algorithm for this, we will set

- `x1` and `x2` to be vectors of (one-dimensional) quadrature of $N+1$ points.
- `y` to be a **two-dimensional** array of $(N+1)^2$ quadrature values. That is, we will set
  `y[i][j] = f(x1[i], x2[j]);`

Derivation

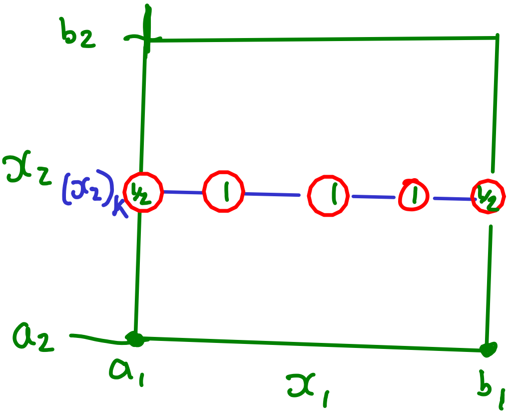
Recall trapezium Rule in 1D

$$\int_a^b f(x)\,dx \approx (b-a)\left(\frac{1}{2}f(x_0) + \sum_{i=1}^{N} f(x_i) + \frac{1}{2}f(x_N)\right)$$



Derivation

In 2D



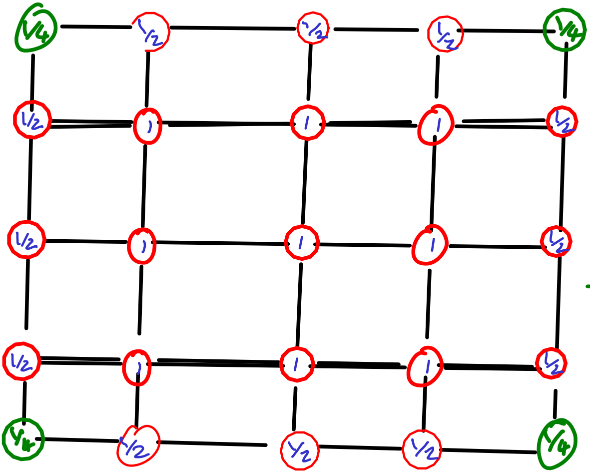
$$\int f(x_1, (x_2)_k)\,dx$$

$$\approx (b_1 - a_1)\Big(\tfrac{1}{2} f((x_1)_0, (x_2)_k)$$

$$+ \sum_{i=1:N} f((x_1)_i, (x_2)_k)$$

$$+ \tfrac{1}{2} f((x_1)_N, (x_2)_k)\Big)$$

Derivation



Method:

$$\int_{a_1}^{b_1}\int_{a_2}^{b_2} f(x_1, x_2)\, dx_1\, dx_2$$

$$\approx (b_1 - a_1)(b_2 - a_2)$$

$\frac{1}{4}$("f at corners")

$+\frac{1}{2}$("f at edges")

$+$ ("f at other points")

Implementation

We'll implement this for estimating $\int_0^1 \int_0^1 e^{x_1+x_2}\,dx_1 dx_2$, with N quadrature points in each direction.

00Trap2D.cpp preamble

```
double f(double x1, double x2) {  return(exp(x1+x2)); }
double ans_true = pow(exp(1.0)-1.0,2); // true value

double Trap2D(double *x1, double *x2,
              double **y, unsigned int N);
```

`00Trap2D.cpp` main()

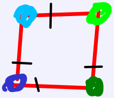
```
int main(void )
{
  unsigned N = pow(2,4);  // Number of points in each direction
  double a1=0.0, b1=1.0, a2=0.0, b2=1.0; // limits of int
  double h1, h2; // step-size in x1 and x2
  double *x1, *x2, **y; // quadrature points and values.

  x1 = new double[N+1];
  x2 = new double[N+1];

  h1 = (b1-a1)/double(N);
  h2 = (b2-a2)/double(N);
  for(unsigned i = 0; i < N+1; i++)
  {
    x1[i] = a1+i*h1;
    x2[i] = a2+i*h2;
  }
```

00Trap2D.cpp main() continued

```
  y = new double * [N+1];
  for(unsigned i = 0; i < N+1; i++)
    y[i] = new double[N+1];

  for (unsigned i=0; i<N+1; i++)
    for (unsigned j=0; j<N+1; j++)
      y[i][j] = f(x1[i], x2[j]);

  double est1 = Trap2D(x1, x2, y, N);
  double error1 = fabs(ans_true - est1);

  std::cout << "N=" <<  N << " | est=" << est1
            << " | error = " << error1 << std::endl;
```

2D - DMA.

00Trap2D.cpp Trap2D()



```
double Trap2D(double *x1, double *x2, double **y,
              unsigned N)
{
  double Q, h1 = (x1[N]-x1[0])/double(N),
    h2 = (x2[N]-x2[0])/double(N);

  Q = 0.25*(f(x1[0],x2[0]) + f(x1[N],x2[0])   // 4 corners
        + f(x1[0],x2[N]) + f(x1[N],x2[N]));

  for (unsigned k=1; k<N; k++)   // 4 edges (not including corners)
    Q += 0.5*(f(x1[k],x2[0]) + f(x1[k],x2[N])
           + f(x1[0],x2[k]) + f(x1[N],x2[k]));

  for (unsigned i=1; i<N; i++) // All the points in the interior
    for (unsigned j=1; j<N; j++)
      Q += f(x1[i],x2[j]);

  Q *= h1*h2;
  return(Q);
```

Lab 6 preview

- Implement Simpson's Rule in 1D and 2D;
- Verify convergence using Python/NumPy/Jupyter.
- Compare with Monte Carlo(?)

Finished here at 10am