Homework #4

Haylee Crane
Paul English

July 23, 2014

1. **(Iterative Solution of Linear Systems.)** Consider the $n \times n$ linear system $Ax = b$ and recall the three iterative techniques we discussed in class:

   Gauss-Jacobi:

$$x_i^{[k+1]} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right)$$

   Gauss-Seidel:

$$x_i^{[k+1]} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right)$$

   Successive Overrelaxation:

$$x_i^{[k+1]} = (1-w) x_i^{[k]} + \frac{w}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right)$$

   As we discussed in class, think of A as being split as

$$A = L + D + U$$

   where L contains the entries of A below the diagonal, U contains the entries above the diagonal, and D contains the entries on the diagonal. For example,

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = L + D + U = \begin{bmatrix} 0 & 0 \\ 3 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$$

   Write the above three iterations as

$$x^{[k+1]} = T_x^{[k]} + c$$

and for each of the three methods give T in terms of L, D, and U, and c in terms of L, D, U and b.

Note: in class we derived the correct expressions for the Gauss-Jacobi method. There was a mistake in the in-class discussion of the Gauss-Seidel method, and I left the SOR method as an exercise.

Gauss-Jacobi:

$$\begin{aligned} x_i^{[k+1]} &= \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \\ x^{[k+1]} &= D^{-1} \left(b - L x^{[k]} - U x^{[k]} \right) \\ &= D^{-1} \left(b - (L+U) x^{[k]} \right) \\ &= -D^{-1} (L+U) x^{[k]} + D^{-1} b \end{aligned}$$

$$\begin{aligned} \Rightarrow T &= -D^{-1} (L+U) \\ c &= D^{-1} b \end{aligned}$$

Gauss-Seidel:

$$\begin{aligned} x_i^{[k+1]} &= \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij} x_j^{[k]} \right) \\ x^{[k+1]} &= D^{-1} \left(b - L x^{[k+1]} - U x^{[k]} \right) \\ x^{[k+1]} &= D^{-1} b - D^{-1} L x^{[k+1]} - D^{-1} U x^{[k]} \\ (I + D^{-1} L) x^{[k+1]} &= D^{-1} b - D^{-1} U x^{[k]} \\ x^{[k+1]} &= -(I + D^{-1} L)^{-1} D^{-1} U x^{[k]} + (I + D^{-1} L)^{-1} D^{-1} b \end{aligned}$$

$$\begin{aligned} \Rightarrow T &= -(I + D^{-1} L)^{-1} D^{-1} U \\ c &= (I + D^{-1} L)^{-1} D^{-1} b \end{aligned}$$

Successive Overrelaxation:

$$
\begin{aligned}
x_i^{[k+1]} &= (1-w)x_i^{[k]} + \frac{w}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{[k+1]} - \sum_{j=i+1}^{n} a_{ij}x_j^{[k]}\right) \\
x^{[k+1]} &= (1-\omega)\,x^{[k]} + \omega D^{-1}\left(b - Lx^{[k+1]} - Ux^{[k]}\right) \\
&= (1-\omega)\,x^{[k]} + \omega D^{-1}b - \omega D^{-1}Lx^{[k+1]} - \omega D^{-1}Ux^{[k]} \\
\left(I + \omega D^{-1}L\right)x^{[k+1]} &= \left((1-\omega) - \omega D^{-1}U\right)x^{[k]} + \omega D^{-1}b \\
x^{[k+1]} &= \left(I + \omega D^{-1}L\right)^{-1}\left((1-\omega) - \omega D^{-1}U\right)x^{[k]} + \left(I + \omega D^{-1}L\right)^{-1}\omega D^{-1}b
\end{aligned}
$$

$$
\begin{aligned}
\Rightarrow T &= \left(I + \omega D^{-1}L\right)^{-1}\left((1-\omega) - \omega D^{-1}U\right) \\
c &= \left(I + \omega D^{-1}L\right)^{-1}\omega D^{-1}
\end{aligned}
$$

2. **(Positive Definite Matrices.)** A *principal submatrix* A_I of an $n \times n$ matrix A is obtained by picking a set $I \subset \{1, 2, \ldots, n\}$ and crossing out all rows and columns whose indices are not in I. For example, if

$$
A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}
$$

then the principal submatrix corresponding to the set $\{2, 4\}$ is

$$
A_{\{2,4\}} = \begin{bmatrix} a_{22} & a_{24} \\ a_{42} & a_{44} \end{bmatrix}
$$

Show that every principal submatrix of a positive definite matrix is positive definite.

For any positive definite matrix we can rearrange a column and a row in the same way and preserve the properties positive definiteness and symmetry. So without loss of generality we can consider just the leading principal submatrices, rather than all possible permutations, i.e. We can just look at $A_{\{1\}}, A_{\{1,2\}}, A_{\{1,2,\ldots\}}, \ldots$.

By induction, for the case $n = 1$ our the only principal submatrix $A_{\{1\}}$ is the same as A and is trivial. For $n = 2$ the only principal submatrix will be $A_{\{1\}}$, and since $A_{\{1\}}$ is a one dimension diagonal matrix we know that it's eigenvalue is the element itself, which is positive, since the diagonals of A are positive, thus for $n = 2$ this is still true.

Now assuming we have a positive definite matrix A of size $n \times n$ and that all principle submatrices $A_{\{1\}}, A_{\{1,2\}}, \ldots, A_{\{1,2,\ldots,n-2\}}$ have been proven to be positive definite, we have to show that $A_{\{1,\ldots,n-2\}}$ is also positive definite.

We know that the eigenvalues of $A_{\{1\}}, A_{\{1,2\}}, \ldots, A_{\{1,2,\ldots,n-2\}}$ are all positive, if we assume that the matrix $A_{\{1,2,\ldots,n-1\}}$ is not positive definite, then it must have a non-positive eigenvalue. However if this were the case then the matrix A would also have to contain a non-positive eigenvalue and we would have a contradiction. Therefore $A_{\{1,2,\ldots,n-1\}}$ must also be positive definite.

3. **(The UL factorization.)** Show how to compute the factorization $A = UL$ where U is upper triangular with 1s along the diagonal and L is lower triangular. Show how this relates to a way of solving $Ax = b$ by transforming the system into an equivalent system with a lower triangular matrix. (In other words, show that what we did for the LU factorization also works for a UL factorization.) Note: For the purposes of this exercise you may assume that no pivoting is required. This is of course unrealistic but pivoting would only distract from the point of this exercise (which is that conceptually there is no difference between an LU and a UL factorization).

   Consider the permutation matrix, P s.t. $A^T = PAP^T$ Then, $PP^T = P^TP = I$ Consider $A^T = PAP^T = L_0U_0$, where L_0U_0 is the LU decomposition of A^T If $U = L_0^T$ and $L = U_0^T$, $L_0U_0 = PUP^TPLP^T = PULP^T = PAP^T \Rightarrow A = UL$ Since U and L both have 1's along the diagonals, $Ax = ULx = b$ can be written as $Lx = y$, and $Uy = b$.

4. **(Spectral Radius.)** We saw that the spectral radius of a (square) matrix A never exceeds an induced matrix norm of $\|A\|$. It can be shown that for any particular matrix A one can find a vector norm such that the induced matrix norm of A is arbitrarily close to the spectral radius of A. Does the spectral radius itself define a norm? Why, or why not?

$$\text{Consider } 0 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\text{and } A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$$

Clearly, $A \neq 0$, the eigenvalues of the zero matrix 0 and A are 0 and 0 respectively.

$$\begin{aligned} \Rightarrow \rho(A) &= \rho(0) \\ &= \max\{0, 0\} \end{aligned}$$

By definition, a norm requires the property $\|A\| = 0 \Rightarrow A = 0$ Here, we have $\rho(A) = 0$, but $A \neq 0$ Therefore, ρ is *not* a matrix norm.

5. **(Inequalities are sharp.)** Explain the meaning of

$$\frac{1}{\|A\|\|A^{-1}\|}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|}$$

and show how we derived these inequalities in class.

- For a general matrix A, and $\|\cdot\| = \|\cdot\|_2$, show that there are non-trivial examples (i.e. $x \neq 0 \neq e$) where the right hand inequality is satisfied with equality in (3). Do the same for the left hand inequality in (3).
- Repeat part a. for $\|\cdot\| = \|\cdot\|_\infty$.

$\frac{\|e\|}{\|x\|}$ is the relative error of our linear system, which may not be computable. $\frac{\|r\|}{\|b\|}$ is a computable value that we can combine with the condition number $\|A\|\|A^{-1}\|$ to estimate what the relative error of our system is.

We say that our system is ill-conditioned if it has a large value for the condition number, which would mean that the above inequality doesn't confine the relative error very well. It's well-conditioned system if the condition number is low, and we can see that the inequality will give us a narrow bounds and a better estimation of the relative error.

To derive these equations we have the following equations from our linear system,

$$\begin{aligned} Ax &= b \\ A^{-1}b &= x \\ Ae &= r \\ A^{-1}r &= e \end{aligned}$$

Where e is our error $x - \hat{x}$ therefore r is a relation of the error and our system.

Using the Cauchy-Schwarz inequality we know that the norms of the above equations follow these inequalities,

$$\begin{aligned} \|b\| &\leq \|A\|\|x\| \\ \|x\| &\leq \|A^{-1}\|\|b\| \\ \|r\| &\leq \|A\|\|e\| \\ \|e\| &\leq \|A^{-1}\|\|r\| \end{aligned}$$

We can divide the first inequality from the fourth and the second from the third (we take the reciprocal reversing the inequality and multiply) to arrive at

$$\frac{\|e\|}{\|A\|\|x\|} \leq \frac{\|A^{-1}\|\|r\|}{\|b\|}$$
$$\frac{\|r\|}{\|A^{-1}\|\|b\|} \leq \frac{\|A\|\|e\|}{\|x\|}$$

Now with the new set of inequalities we multiply the first by $\|A\|$ and the second by $\frac{1}{\|A\|}$ to get to the original inequality,

$$\frac{1}{\|A\|\|A^{-1}\|}\frac{\|r\|}{\|b\|} \leq \frac{\|A\|\|e\|}{\|A\|\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|}$$
$$\Rightarrow \frac{1}{\|A\|\|A^{-1}\|}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \|A\|\|A^{-1}\|\frac{\|r\|}{\|b\|}$$

For $\|\cdot\| = \|\cdot\|_2$ we have that $\|A\| = \max_{Ax=\lambda x}\{\lambda\} = \sqrt{\lambda_1}$ or the square root of the largest eigenvalue, thus the norm of A^{-1} will be one over the square root of the smallest eigenvalue $\|A^{-1}\| = \frac{1}{\min_{Ax=\lambda x}\{\lambda\}} = \sqrt{\frac{1}{\lambda_n}}$ since the eigenvalues of an inverted matrix are reciprocals. Now our condition number is $\|A\|\|A^{-1}\| = \sqrt{\frac{\lambda_1}{\lambda_n}}$, and our relative error is bounded by,

$$\sqrt{\frac{\lambda_n}{\lambda_1}}\frac{\|r\|}{\|b\|} \leq \frac{\|e\|}{\|x\|} \leq \sqrt{\frac{\lambda_1}{\lambda_n}}\frac{\|r\|}{\|b\|}$$

Here inequality is achieved when $\lambda_1 = \lambda_n$. We can find this case trivially with the identity matrix, i.e. $A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$.

With $\|\cdot\| = \|\cdot\|_\infty$ our matrix norm is $\|A\| = \max\{\|a_{ij}\|\} = a_{\max}$ and $\|A^{-1}\| = \frac{1}{\min\{\|a_{ij}\|\}} = \frac{1}{a_{\min}}$. This means that our condition number is $\|A\|\|A^{-1}\| = \frac{a_{\max}}{a_{\min}}$, and our inequality will have the case of equality when $a_{\max} = a_{\min}$. An example of this is $A = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$.

6. **(Backward Error Analysis.)** This problem explores the effects of a perturbation in the coefficient matrix (rather than the right hand side) of the linear system

$$Ax = b$$

Suppose we solve instead of (4) the system

$$(A - E)(x - e) = b$$

where E is a perturpation of A that causes an error e in the solution x. Show that

$$\frac{\|e\|}{\|x - e\|} \leq \|A\|\|A^{-1}\|\frac{\|E\|}{\|A\|}$$

$$\begin{aligned}
Ax &= b \\
Ae &= r, e = A^{-1} \\
(A - E)(x - e) &= b \\
\Rightarrow A(x - e) - E(x - e) &= b \\
\Rightarrow Ax - Ae - b &= E(x - e) \\
\Rightarrow -r &= E(x - e) \\
\|e\| &\leqslant \left\|A^{-1}\right\| \|r\| \\
\|r\| &\leqslant \|E\| \|(x - e)\| \\
\Rightarrow \frac{\|e\|}{\|E\| \|(x - e)\|} &\leqslant \frac{\left\|A^{-1}\right\| \|r\|}{\|r\|} \\
\Rightarrow \frac{\|e\|}{\|(x - e)\|} &\leqslant \left\|A^{-1}\right\| \|E\| = \|A\| \left\|A^{-1}\right\| \frac{\|E\|}{\|A\|} \\
\Rightarrow \frac{\|e\|}{\|(x - e)\|} &\leqslant \|A\| \left\|A^{-1}\right\| \frac{\|E\|}{\|A\|}
\end{aligned}$$

7. **(First Order Systems.)** Write the second order initial value problem

$$y'' = xy^2 \tag{1}$$
$$y(0) = 1 \tag{2}$$
$$y'(0) = 2 \tag{3}$$

as an autonomous first order system

$$y' = f(y) \tag{4}$$

$$y(a) = y_0. \tag{5}$$

(In other words, specify y, f, a, and y_0 such that the two problems are equivalent. Of course, y will have different meanings for the two problems.)

$$\begin{aligned} y' &= xy^2 \\ y(0) &= 1 \\ y'(0) &= 2 \end{aligned}$$

Let $z = y'$ and $z(0) = 2$

$$\Rightarrow z' = xy^2, y(0) = 1, z(0) = 2$$

$$\begin{aligned} \frac{\partial y}{\partial x} &= z, y(0) = 1 \\ \frac{\partial z}{\partial x} &= xy^2, z(0) = 2 \end{aligned}$$

Define $x = x(t)$ s.t. $\frac{\partial x}{\partial t} = 1$ and $x(0) = 0$

$$\begin{aligned} \Rightarrow \frac{\partial y}{\partial x} &= \frac{\partial y}{\partial x}\frac{\partial x}{\partial t} = z \\ \frac{\partial z}{\partial t} &= \frac{\partial z}{\partial x}\frac{\partial x}{\partial t} = x(t)y^2(t) \\ \Rightarrow \frac{\partial x}{\partial t} &= 1, x(0) = 0 \\ \frac{\partial y}{\partial t} &= z(t), y(x(0)) = 1 \\ \frac{\partial z}{\partial t} &= x(t)y^2(t), z(x(0)) = 2 \\ v &= \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix}, f(v) = \begin{bmatrix} v \\ ve_3 \\ (ve_1)(ve_2)^2 \end{bmatrix} \end{aligned}$$

Where e_1, e_2, e_3 are the basis vectors

$$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix}$$

respectively

$$\frac{\partial v}{\partial t} = f(v), v(0) = \begin{bmatrix}0\\1\\2\end{bmatrix}$$

8. **(Improper Integrals.)** Let

$$I = \int_{-1}^{1} \frac{\ln(x^2+1)}{\sqrt{1-x^2}} dx = 2\pi \ln\left[\frac{1+\sqrt{2}}{2}\right].$$

This integral is improper because the integrand approaches infinity as x approaches ± 1. Attempt to approximate this integral by using Simpson's Rule (or any other Newton-Cotes formula) on the interval $[-1+\epsilon, 1-\epsilon]$ for small $\epsilon > 0$. Describe the results of your efforts. Then, for $n = 4, 5, 6, 7, 8$, use the Gaussian Quadrature formula

$$\int_{-1}^{1} \frac{f(x)}{\sqrt{1-x^2}} dx = \frac{\pi}{n} \sum_{i=1}^{n} f\left(\cos\left(\frac{(2i-1)\pi}{2n}\right)\right)$$

which we discussed in class. Compute the error and discuss your results.

```clojure
(def I (* 2 Math/PI
          (Math/log (/ (+ 1 (Math/sqrt 2))
                       2))))

(defn gaussian-quadrature [f n]
  (* (/ Math/PI n)
     (->> (range n)
          (map inc)
          (map #(f (Math/cos (/ (* (- (* 2 %) 1)
                                   Math/PI)
                                (* 2 n)))))
          (apply +))))

(def n (range 4 9))
```

```clojure

(def results (map #(gaussian-quadrature (fn [x]
                                          (Math/log (+ (Math/pow x 2) 1)))
                                        %)
                  n))
;; => (1.1840219784143866
;;     1.1824745448480443
;;     1.182688104009816
;;     1.1826574630224815
;;     1.1826619812548276)

(def error (->> results
                (map #(Math/abs (- I %)))
                (zipmap [4 5 6 7 8])))
;; => {4 0.0013605869242299118,
;;     5 1.8684664211243707E-4,
;;     6 2.6712519659355394E-5,
;;     7 3.928467675162752E-6,
;;     8 5.897646708774573E-7}
```

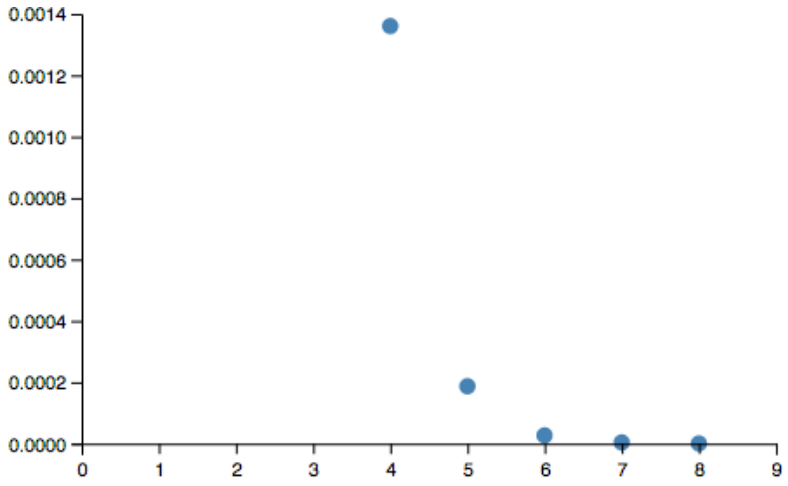

Figure 1: Error for $n = 4, 5, 6, 7, 8$.

We can see that our error decreases for larger values of n and approaches 0 quickly. Even for $n = 4$ it's accurate.

9. **(Numerical Comparison.)** For Euler's Method, the Trapezoidal Rule, Simpson's

Rule, and the standard 4-th order Runge-Kutta Method solve the initial value problem

$$y' = y \tag{6}$$
$$y(0) = 1 \tag{7}$$
$$x \in [0, 1] \tag{8}$$

with step-sizes $h = 2^{-s}$ where $s = 3, 4, \ldots, 15$. Plot the error at the right end-point, i.e. $y(1) - y_n$ (where $n = 1/h$), against h. You may wish to superimpose the plots. If you like use appropriate logarithmic graph paper. Comment on your results. (For Simpson's rule use the exact starting value $y_1 = y(h)$.) Send me your program by e-mail.

The "standard" Runge-Kutta method is:

$$y_{n+1} = y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

where

$$\begin{aligned} K_1 &= f(x_n, y_n) \\ K_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_1\right) \\ K_3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}K_2\right) \\ K_4 &= f\left(x_n + h, y_n + hK_3\right) \end{aligned}$$

First we define a general method for iteration. Our iteration method takes a function of the form $y'(x, y)$, and in our case we just return y. Next we define a list of the step sizes we would like to compare against.

```
(defn lmm-ode-solve [f init & {:keys [x-max h method]}]
  {:pre [x-max h method]}
  (take-while #(<= (first %) x-max)
              (iterate (method f h)
                       init)))

(def y' (fn [x y] y))
(def s (range 3 16))
(def h (map #(Math/pow 2 (- %)) s))
```

Each method function returns a method that can be iterated on. This lets us encapsulate the original ODE, and step size. We return a vector with $[x, y]$ values so that the next iteration can make use of both. The final result is a list of coordinate pairs which we can plot or use to measure error. We will use `map` to run this on all of our step sizes.

(a) **Euler's Method**

$$y_{n+1} - y_n = hf_n \tag{9}$$

```
(defn eulers [f h]
  (fn [[x y]]
    [(+ x h)
     (+ y (* h (f x y)))]))

(def euler-results (map #(lmm-ode-solve y'
                                        [0 1]
                                        :x-max 1
                                        :h %
                                        :method eulers)
                        h))
```

(b) **Trapezoidal Rule**

$$y_{n+1} - y_n = \frac{h}{2}(f_{n+1} + f_n) \tag{10}$$

```
(defn trapezoidal-rule [f h]
  (fn [[x y]]
    (let [K1 (* h (f x y))
          K2 (* h (f (+ x h) (+ y K1)))]
      [(+ x h)
       (+ y (* 1/2 (+ K1 K2)))])))

(def trapezoid-results (map #(lmm-ode-solve y'
                                            [0 1]
                                            :x-max 1
                                            :h %
                                            :method trapezoidal-rule)
                            h))
```

(c) **Simpons's Rule**

$$y_{n+2} - y_n = \frac{h}{3}(f_{n+2} + 4f_{n+1} + f_n) \tag{11}$$

```
(defn simpsons-rule [f h]
  (fn [[x y]]
    (let [K1 (* h (f x y))
          K2 (* h (f (+ x (/ h 2)) (+ y K1)))
          K3 (* h (f (+ x h) (+ y K2)))]
      [(+ x (* h 2))
       (+ y (* 1/3 (+ K1 (* 4 K2) K3)))])))

(def simpsons-results (map #(lmm-ode-solve y'
                                           [0 1]
                                           :x-max 1
                                           :h %
                                           :method simpsons-rule)
                           h))
```

(d) **Runge Kutta:**

```
(defn runge-kutta-4 [f h]
  (fn [[x y]]
    (let [K1 (* h (f x y))
          K2 (* h (f (+ x (/ h 2)) (+ y (/ K1 2))))
          K3 (* h (f (+ x (/ h 2)) (+ y (/ K2 2))))
          K4 (* h (f (+ x h) (+ y K1)))]
      [(+ x h)
       (+ y (* 1/6 (+ K1 (* 2 K2) (* 2 K3) K4)))])))

(def runge-results (map #(lmm-ode-solve y'
                                        [0 1]
                                        :x-max 1
                                        :h %
                                        :method runge-kutta-4)
                        h))
```

Lastly we can compute the error values and plot them.

```clojure
(defn errors [results]
  (map #(Math/abs (- (y 1) (last (last %))))
       results))

(def euler-errors (errors euler-results))
(def trapezoid-errors (errors trapezoid-results))
(def simpsons-errors (errors simpsons-results))
(def runge-errors (errors runge-results))
```

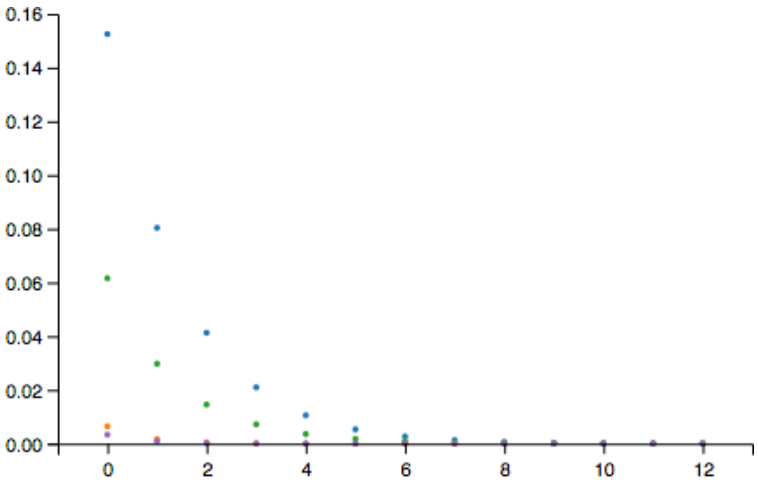

Figure 2: Error for each value of h. The indices of the x-axis are shifted: 0 is for $h = 2^{-3}$, 1 for $h = 2^{-4}$, ... Blue is Euler's method, orange is the trapezoid rule, green is Simpson's Rule, and purple is the Runge Kutta method.

We can see that each method performs differently, but that they all achieve more accuracy with a smaller step size. In particular the Runge-Kutta method and the trapezoid rule are very effective.